НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Автоматизованих систем обробки інформації і управління
(повна назва кафедри)

«На правах рукопису»
УДК 004.94

До захисту допущено:
В.о. завідувача кафедри
_____ Олександр ПАВЛОВ
«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютеризованих систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Сервіс паралельної імітації дискретно-подійних систем»

Виконав (-ла):
студент (-ка) VI курсу, групи ІП-92мп
Педоренко Андрій Вікторович _____

Науковий керівник:
Проф., д.т.н.,
Стеценко Інна Вячеславівна _____

Рецензент:
Доцент, д.т.н.,
Клименко Ірина Анатоліївна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Автоматизованих систем обробки інформації і управління

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма - «Інженерія програмного забезпечення комп'ютеризованих систем»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр ПАВЛОВ
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Педоренку Андрію Вікторовичу

1. Тема дисертації «Сервіс паралельної імітації дискретно-подійних систем», науковий керівник дисертації Стеценко Інна Вячеславівна, проф., д.т.н., затверджені наказом по університету від «26» жовтня 2020 р. №3132-с

2. Термін подання студентом дисертації _____

3. Об'єкт дослідження Алгоритм паралельної імітації дискретно-подійних моделей

4. Вхідні дані Технічне завдання

5. Перелік завдань, які потрібно розробити аналіз формалізмів дискретно-подійних систем; порівняння підходів до архітектури систем; дослідження паралельного алгоритму імітації Петрі-об'єктних моделей; моделювання та конструювання програмного забезпечення; розробка бізнес-плану проєкту.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу

1) Схема структурна варіантів використань

2) Схема структурна класів вузлового сервісу

7. Орієнтовний перелік публікацій 1 публікація

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Графічний	доц. Ліщук К.І.		

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Вивчення рекомендованої літератури	04.09.2020	
2	Аналіз формалізмів дискретно-подійних систем	04.09.2020	
3	Аналіз підходів до архітектури	04.09.2020	
4	Постановка та формалізація задачі	11.09.2020	
5	Дослідження існуючих паралельних алгоритмів імітації	25.09.2020	
6	Моделювання програмного забезпечення	23.10.2020	
7	Розробка програмного забезпечення	13.11.2020	
8	Виконання графічних документів	25.11.2020	
9	Оформлення дисертації	25.11.2020	
10	Подання МД на попередній захист	27.11.2020	
11	Подання МД рецензенту		
12	Подання МД на основний захист	17.12.2020	

Студент Педоренко А.В.

Науковий керівник Стеценко І.В.

РЕФЕРАТ

Актуальність теми: Процеси та системи стають все більшими та складнішими. При неправильній побудові процесів втрачається велика кількість коштів, тому їх необхідно моделювати перед створенням. Імітація таких великих моделей займає багато часу, а для досконального дослідження імітацію треба проводити декілька разів, та ще й з різними параметрами. Пришвидшити імітацію можна за допомогою паралельних обчислень. Крім того, на надвеликих моделях виникають труднощі при імітації на одному комп'ютері/сервері, що можна вирішити за допомогою розподілених обчислень.

Мета дослідження: Підвищення ефективності паралельної реалізації алгоритму імітації Петрі-об'єктних моделей для надвеликих моделей.

Завдання дослідження:

- аналіз формалізмів дискретно-подійних систем;
- порівняння підходів до архітектури систем;
- дослідження паралельного алгоритму імітації Петрі-об'єктних моделей;
- розробка розподіленого алгоритму імітації Петрі-об'єктних моделей;
- моделювання та конструювання сервісу імітації.

Об'єкт дослідження: Паралельний алгоритм імітації Петрі-об'єктних моделей.

Предмет дослідження: Ефективність паралельного алгоритму імітації Петрі-об'єктних моделей для надвеликих моделей.

Наукова новизна:

Удосконалено паралельний алгоритм імітації Петрі-об'єктних моделей для надвеликих моделей за рахунок впровадження його в розподіленій обчислювальній системі.

Практичне значення отриманих результатів визначається тим, що отриманий розподілений алгоритм можна впроваджувати в програмне

забезпечення імітації дискретно-подійних систем, що зменшить час імітації та дозволить імітувати надвеликі моделі.

Зв’язок з науковими програмами, планами, темами: робота виконувалась на кафедрі автоматизованих систем обробки інформації і управління Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського".

Публікації: наукові положення дисертації опубліковано в Педоренко А.В. Стеценко І.В. Паралельний алгоритм обчислень Петрі-об’єктних моделей / Педоренко А.В., Стеценко І.В // Матеріали V Всеукраїнської науково-практичної конференції молодих вчених та студентів «Інформаційні системи та технології управління»(ІСТУ-2020) - м. Київ: НТУУ “КПІ ім. Ігоря Сікорського”, 26-27 листопада 2020р.

КЛЮЧОВІ СЛОВА: СТОХАСТИЧНІ МЕРЕЖІ ПЕТРІ, ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ, РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ, АЛГОРИТМ ІМІТАЦІЇ

ABSTRACT

Topicality: Processes and systems are becoming larger and more complex. Improper construction of processes loses a lot of money, so they must be modeled before creation. Simulation of such large models takes a long time, and for a thorough study, the simulation must be performed several times, and even with different parameters. You can speed up the simulation with the help of parallel calculations. In addition, ultra-large models have difficulty simulating on a single computer / server, which can be solved using distributed computing.

The aim of the study: Increasing the efficiency of parallel implementation of the algorithm for simulating Petri-object models for ultra-large models.

Tasks of the study:

- analysis of formalisms of discrete-event systems;
- comparison of approaches to system architecture;
- research of parallel algorithm of simulation of Petri-object models;
- development of a distributed algorithm for simulating Petri-object models;
- modeling and construction of simulation service.

Object of study: The parallel algorithm for simulating Petri object models.

Subject of research: Efficiency of the parallel algorithm for simulating Petri-object model for ultra-large models.

Scientific novelty:

Improved the parallel algorithm for simulating Petri-object models for ultra-large models with implementation it in a distributed computing system.

The practical value of the obtained results is determined by the fact that the obtained distributed algorithm can be implemented in the software for simulation of discrete-event systems, which will reduce the simulation time and allow to simulate ultra-large models.

Relationship with scientific programs, plans and themes: The work was carried out at the Department of Automated Information Processing and Management

Systems of the National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

Publications: Scientific provisions of the dissertation published in Pedorenko A.V. Stetsenko I.V A Parallel Algorithm for Computing Petri-Object Models / Pedorenko A.V., Stetsenko I.V // Proceedings of the Fifth All-Ukrainian Scientific and Practical Conference of Young Scientists and Students "Information Systems and Management Technologies" (ISTU-2020) - Kyiv: NTUU “KPI them. Igor Sikorsky”, November 26-27, 2020.

KEY WORDS: STOCHASTIC PETRI NET, PARALLEL CALCULATIONS, DISTRIBUTED COMPUTING, SIMULATION ALGORITHM

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AMQP – Advanced message queuing protocol

API – Application programming interface

CRUD – Create, remove, update, delete

DDD – Domain-driven design

JMS – Java message service

HTTP – Hypertext transfer protocol

MSMQ – Microsoft message queuing

REST – Representational state transfer

SOA – Service-oriented architecture

SOAP – Simple object access protocol

СМО – Система масового обслуговування

ВСТУП

Тенденція розвитку економіки світу – ускладнення організацій, технологій, процесів, логістики. З появою нових технологічних проривів, що мають облегшувати, автоматизувати та пришвидшувати процеси, структура цих процесів сильно ускладняється. Старі способи налагодження процесів, як еволюційне покращення, коли спостерігається вже існуюча система та покращуються її частини, що працюють найменш ефективно, вже не є дієвими. По-перше, будь-яка зміна в процесах настільки складних, наскільки вони вже є зараз, може привести до неочевидних наслідків. По-друге, побудова процесів надзвичайно вартісна, тому необхідно якнайшвидше отримати максимально ефективний варіант.

Сучасне рішення цієї проблеми – імітація. Процес, що розробляється, моделюють у вигляді дискретно-подійних систем, визначаючи основні елементи, розраховуючи затримки на етапах процесу, налаштовуючи зв’язки між елементами. На отриманій моделі потім можна запустити імітацію за допомогою алгоритму. Дослідникам залишається лише проаналізувати роботу імітації, визначити слабкі частини системи, перебудувати моделі та повторити процес імітації. Виконують ці дії до тих пір, поки не отримають оптимальну модель системи.

Існують інструменти, які дозволяють будувати модель та запускати на ній імітацію, але вони чи не дозволяють з достатнім рівнем гнучкості створювати моделі, чи сильно ускладнюють модель за рахунок величезної кількості різноманітних можливих елементів. Крім того, здебільшого вони встановлюються на ресурси користувача, в той час коли більш ефективно є використання ресурсів постачальника інструменту імітації чи навіть хмарних ресурсів. Також ці інструменти можуть не підходити для надвеликих моделей, оскільки для них потрібно використовувати розподілені алгоритми.

Мета даної роботи – підвищення ефективності паралельної реалізації алгоритму імітації Петрі-об’єктних моделей для надвеликих моделей.

Об’єкт дослідження даної роботи – паралельний алгоритм імітації Петрі-об’єктних моделей.

Предмет дослідження даної роботи – ефективність паралельного алгоритму імітації Петрі-об’єктних моделей для надвеликих моделей.

Наукова новизна даної роботи – удосконалено паралельний алгоритм імітації Петрі-об’єктних моделей для надвеликих моделей за рахунок впровадження його в розподіленій обчислювальній системі.

Завданням даної роботи є розробка сервісу паралельної імітації дискретно-подійних систем, що використовує Петрі-об’єктні моделі у якості формалізму дискретно-подійних систем, виконує імітацію на ресурсах сервісу та спроможна імітувати надвеликі моделі. Результатом роботи є сервіс імітації, розроблений мовами Java та JavaScript.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Дискретно-подійне моделювання

Дискретно-подійне моделювання - це техніка, що використовується для моделювання реальних систем, які можна розбити на набір логічно різних процесів, які з часом прогресують автономно.[1] Кожна подія відбувається в певному процесі і їй призначається логічний час (позначка часу). Впливом цієї події може бути результат, переданий одному або декільком іншим процесам. Зміст результату може призвести до генерації нових подій, які будуть оброблені в якийсь момент у майбутньому. Основна статистична парадигма, яка підтримує дискретні системи подій, базується на теорії черг. Історично такий підхід використовувався для оцінки планування телефонних дзвінків, а останнім часом і розподілу діяльності через комп'ютерну мережу. Корисна модель черги представляє реальну систему з достатньою точністю та аналітично керованою. Модель масового обслуговування, заснована на процесі Пуассона та супутньому експоненціальному розподілі ймовірності, часто задовольняє цим двом вимогам. Процес Пуассона моделює випадкові події (наприклад, прихід клієнта, запит на дії з веб-сервера або завершення дії, що вимагається від веб-сервера) як такі, що виходять із процесу, що обмежений за своєю природою. Отже, тривалість проміжку часу від поточного часу до наступної події не залежить від часу, коли відбулася остання подія. У розподілі Пуассона спостерігач реєструє кількість подій, що відбуваються за інтервал часу фіксованої довжини. При експоненційному (негативному) розподілі ймовірностей спостерігач фіксує тривалість інтервалу часу між послідовними подіями. В обох випадках основний фізичний процес не має пам’яті. Моделі, засновані на процесі Пуассона, часто реагують на входи з навколишнього середовища таким чином, що імітують реакцію змодельованої системи на ті самі входи. Моделі, аналітично сприйнятливі до моделювання, які перетворюються на інформацію про систему, що моделюється, та форму їх

рішення. Модель коригування на основі Пуассона, яка відносно погано імітує детальну роботу системи, також може бути корисною. Той факт, що такі моделі часто надають найгірший сценарій, привернув розробників систем, які воліють включати в свої конструкції коефіцієнт безпеки. Крім того, форма вирішення моделей Пуассона часто надає інформацію про форму вирішення проблеми масового обслуговування, детальна поведінка якої погано моделюється. Отже, моделі масового обслуговування часто моделюються як процеси Пуассона, які використовують експоненціальний розподіл.

Щоб підсумувати, як працює моделювання дискретно-подійних [2] систем (рис. 1.1), наведено чотири кроки:

а) Спостерігається "реальну" динаміку системи (яка майже завжди є безперервною).

б) Враховується лише важливі моменти життя системи і вони розглядаються як миттєві неподільні події. Усі зміни в системі будуть пов'язані з цими подіями.

в) Моделююча програма підтримує чергу подій та серіалізує події; в той же час це забезпечує синхронізацію часу моделі, протягом якого працює програма, з цими реалізованими подіями.

г) Потім симулятор запускає модель, перескакуючи від однієї події до іншої, відстежуючи змодельований час, пов’язаний з подіями.

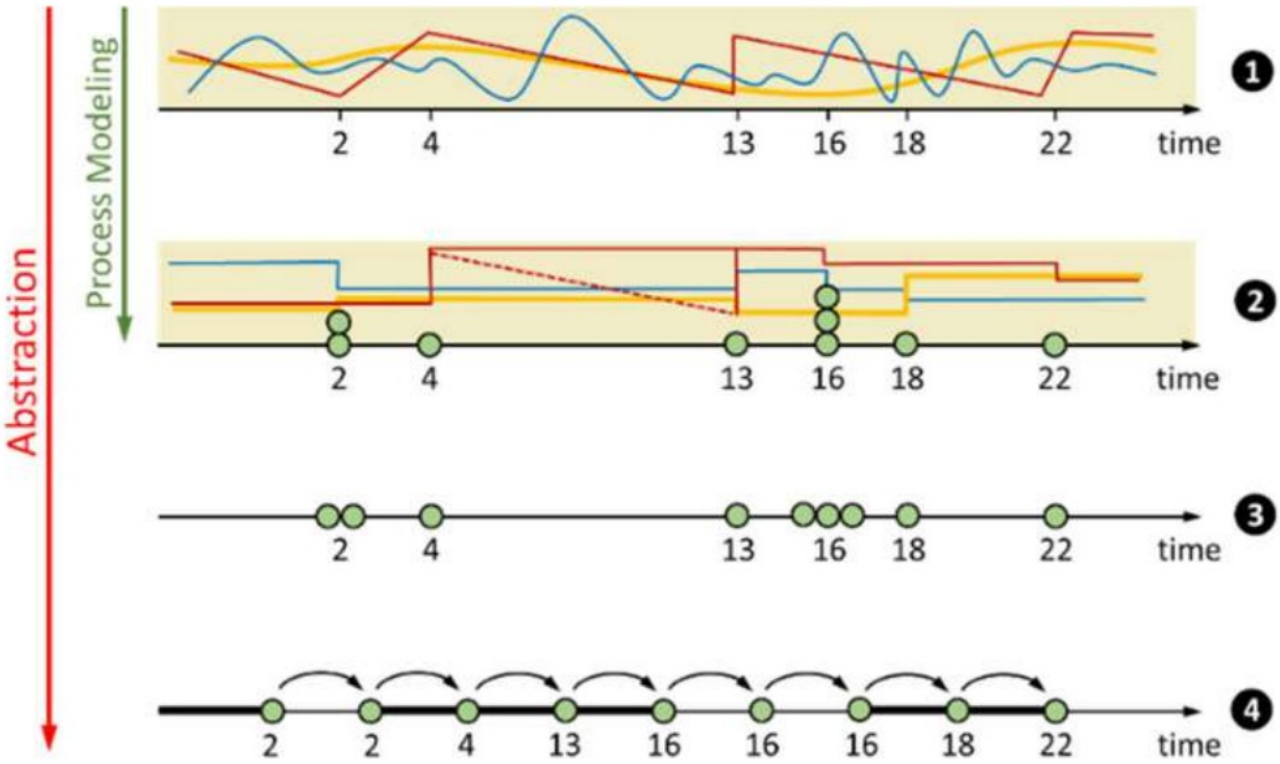


Рисунок 1.1 – Процес моделювання дискретно-подійних систем

1.1.1 Система масового обслуговування

У імітаційній моделі процес-орієнтована абстракція системи може бути змодельована блоками управління потоком. Основна поведінка цього підходу успадковується від розділу математики, який називається теорією масового обслуговування. Хоча теорія масового обслуговування більше зосереджена на математичних рішеннях (тобто аналітичному підході, а не на моделюванні) потокових систем, важливо переглянути її, щоб краще зрозуміти оригінальну методологію, що лежить в основі моделювання, орієнтованого на процес.

Теорія масового обслуговування - це розділ математики, що досліджує черги. Система масового обслуговування - це система, в якій об’єкти (речі, що рухаються по системі) потрапляють, обробляються ресурсом (або декількома ресурсами) із системних ресурсів, а потім виходять із системи або повертаються до попередньої точки. У цьому потоці існує ймовірність того, що кількість об’єктів, які звертаються за обробкою, перевищує доступну. Ця

нестача ресурсів призведе до того, що об’єкти будуть чекати в чергах, перш ніж будуть оброблені.

Оскільки системи масового обслуговування структуровані певним чином, існує стандартний спосіб їх концептуалізації як процесів. У цих системах за допомогою простого процесу можна уявити, як ресурс допомагає об’єктам отримати потрібну послугу. Наприклад: коли пасажир прибуває до столу реєстрації в аеропорту, ресурсом (або "сервером" за термінологією системи масового обслуговування) є представник авіакомпанії, яка приймає багаж. Якщо пасажирів, які бажають зареєструвати свій багаж, більше, ніж представників, буде обслуговуватися лише кількість пасажирів, рівна кількості вільних представників; решта повинні чекати в черзі. Виходячи з цього, простий процес можна змоделювати за допомогою комбінації черги та сервера, пари, відомої як сервісний центр або коротше сервіс (рис. 1.2).

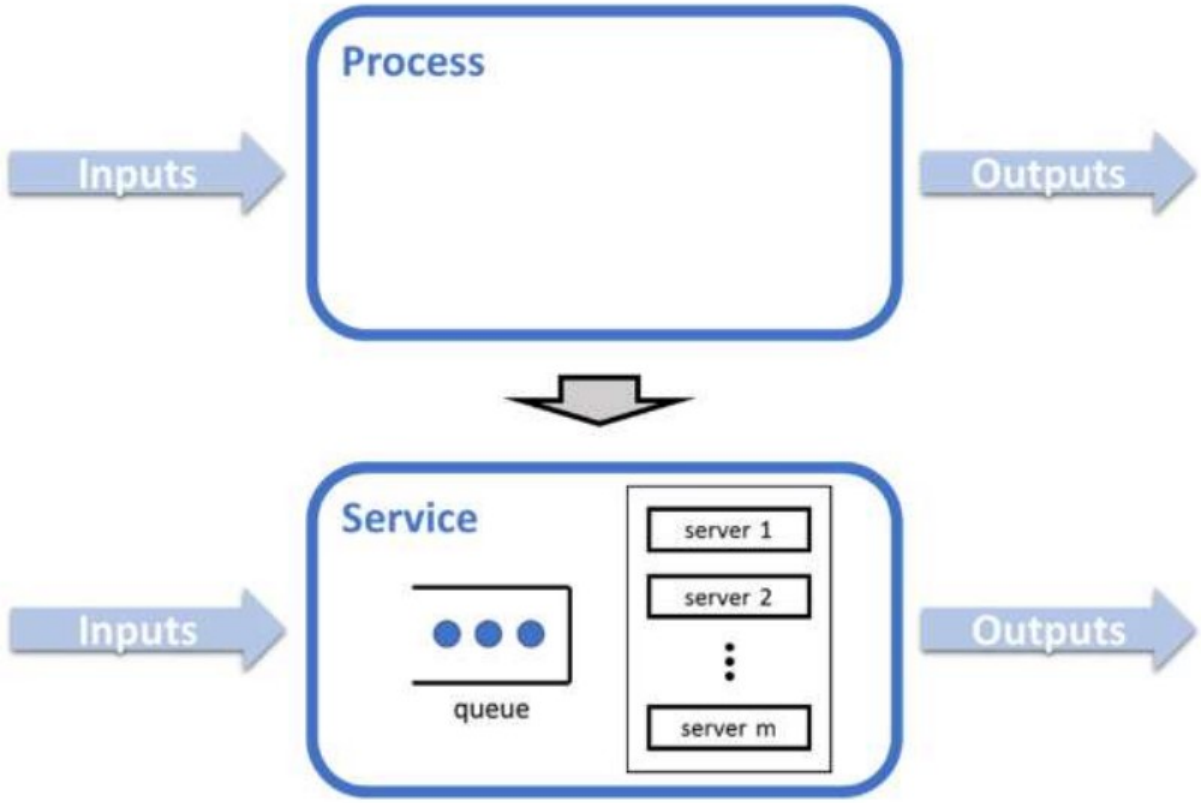


Рисунок 1.2 – Моделювання процесу за допомогою системи масового обслуговування

1.1.2 Мережі Петрі

Мережі Петрі - це графічне зображення системи, в якій одночасно виконуються кілька незалежних дій [3]. Можливість моделювання декількох дій відрізняє мережі Петрі від кінцевих автоматів. У кінцевих автоматів завжди існує "поточний" стан, який визначає, яка дія може відбуватися наступним чином. Мережі Петрі можуть мати кілька станів, кожен з яких може розвиватися, змінюючи стан мережі Петрі. Як варіант, деякі з цих станів можуть розвиватися паралельно, що спричиняє одночасне відбування кількох незалежних змін у мережі Петрі.

Мережа Петрі складається з позицій, переходів та дуг, що їх з'єднують. Вхідні дуги з'єднують позиції з переходами, а вихідні дуги починаються на переході і закінчуються в позиції. Позиції можуть містити маркери; Поточний стан змодельованої системи (нумерація) задається кількістю (та типом, якщо маркери можна розрізнити) маркерів у кожному положенні. Переходи є активними компонентами. Вони імітують дії, які можуть відбутися (ініціюється перехід), змінюючи тим самим стан системи (маркування мережі Петрі). Переходи можна активувати, лише якщо вони ввімкнені, а це означає, що всі передумови для дії повинні бути виконані (у вхідних позиціях доступно достатньо маркерів). Коли перехід спрацьовує, він видаляє маркери зі своїх вхідних позицій і додає до вихідних позицій. Кількість видалених / доданих маркерів залежить від кратності кожної дуги.

Базова мережа Петрі зображена на рисунку 1.3. Ця мережа Петрі має чотири позиції, позначені P0 до P4, і три переходи, позначені T0 до T2. Кожна позиція P0 і P2 має маркер, представлений чорною крапкою в кожній позиції. Дуги, представлені у вигляді спрямованих дуг, з'єднують позиції з переходами, а переходи з позиціями. У добре сформованій мережі Петрі позиції не можуть бути безпосередньо пов'язані з іншими позиціями, а переходи не можуть бути безпосередньо пов'язані з іншими переходами. Також зверніть увагу, що мережа Петрі може містити петлі. Мережа Петрі на рисунку 1.3 містить дві

петлі. Один цикл містить P0, T0, P1, T1, P3 і T3. Інший цикл містить T1, P4, T2 і P2. У мережах Петрі часто є петлі, що представляють повторювані дії. Наприклад, веб-сервер знову обслуговує вхідні запити для видачі вмісту веб-сторінок різними клієнтами.

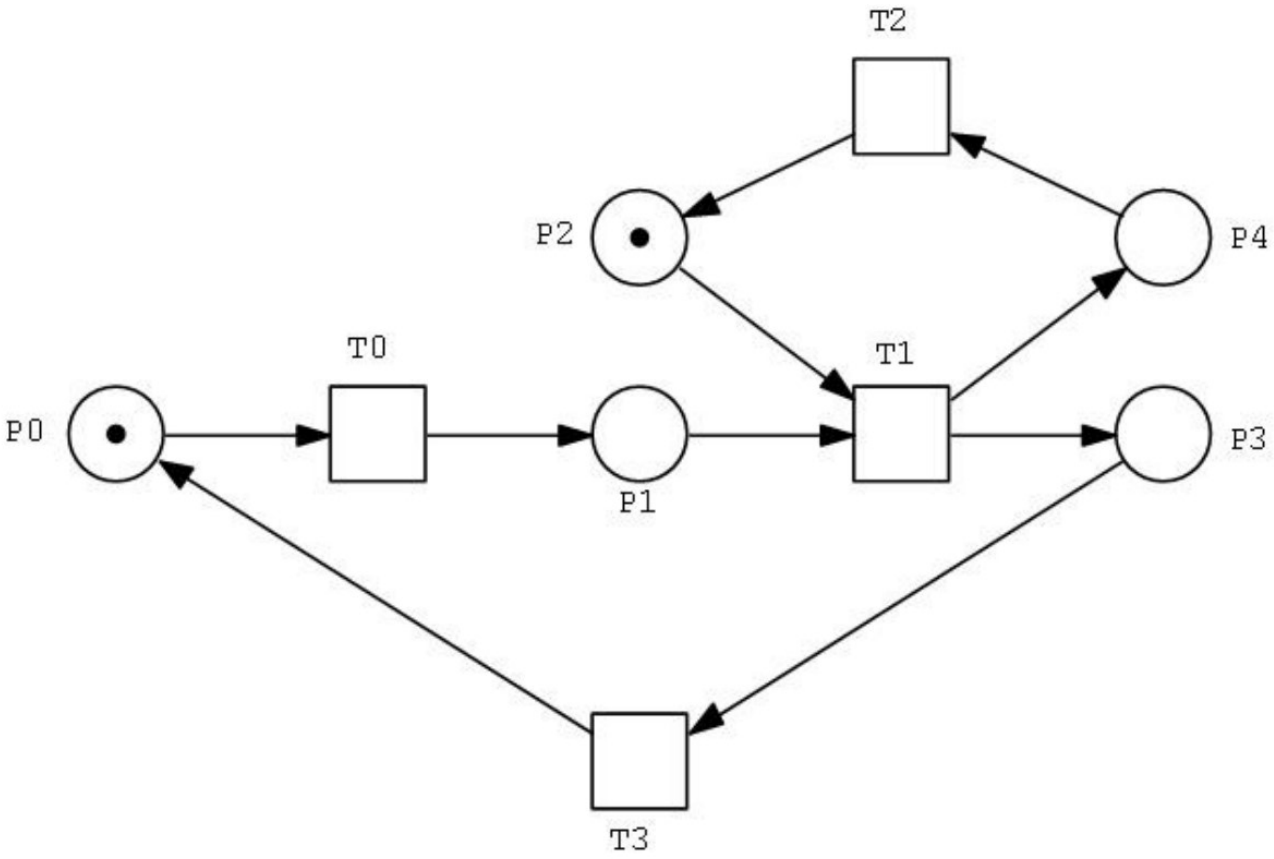


Рисунок 1.3 – Приклад мережі Петрі

Стан мережі Петрі представлений наявністю маркерів у різних позиціях. Стан мережі Петрі на рисунку 1.3 містить маркери в позиціях P0 і P2. В іншому стані цієї мережі Петрі будуть маркерив позиціях P1 і P2. В іншому стані маркери перейдуть в позиції P3 і P4. Не всі маркери на позиціях представляють можливий стан системи. Наприклад, мережа Петрі на рисунку 1.3 ніколи не матиме, як можливий стан, такий стан, в якому єдині маркери знаходяться в позиціях P1 і P4. Які стани можливі, а які ні, визначаються структурою мережі Петрі та правилами, що визначають, як мережа Петрі змінює свій стан.

Мережі Петрі є перспективним інструментом для опису та вивчення систем, що характеризуються як паралельні, асинхронні, розподілені,

паралельні, недетерміновані та / або стохастичні. Як графічний інструмент, мережі Петрі можна використовувати як інструмент візуального спілкування, подібний до блок-схем та мереж. Крім того, ці мережі використовують маркери для імітації активності динамічної та паралельної системи. Як математичний інструмент існує можливість створювати рівняння стану, алгебраїчні рівняння та інші математичні моделі, що регулюють поведінку систем.

Для вивчення характеристик та надійності систем необхідно включити в модель поняття часу. Є кілька способів зробити це в мережі Петрі; однак, найпоширеніший спосіб - пов'язати затримку активації з кожним переходом. Ця затримка визначає, як довго перехід повинен бути активований, перш ніж він насправді активується. Якщо затримка є функцією випадкового розподілу, результуючий клас мережі називається стохастичною мережею Петрі. Залежно від асоційованої латентності можна виділити різні типи переходів, такі як негайні переходи (без затримки), експоненційні переходи (затримка є експоненціальним розподілом) та детерміновані переходи (затримка фіксована).

1.2 Архітектурні підходи розробки програмного забезпечення

1.2.1 Монолітна архітектура

Монолітний додаток побудований як єдиний блок, де інтерфейс користувача та код доступу до даних поєднуються в єдину програму на одній платформі [4]. Монолітні бізнес-додатки складаються з трьох частин (рисунок 1.4): база даних, що складається з багатьох таблиць, як правило, в реляційній системі управління базами даних [5]. Клієнтський інтерфейс, що складається з HTML та/або JavaScript, що працює в браузері. Серверні програми, які виступають посередником між інтерфейсом користувача та базою даних, оброблятимуть HTTP-запити, виконуватимуть логіку для конкретного домену, отримуватимуть та оновлюватимуть дані з бази даних, а також заповнюватимуть подання HTML для надсилання до браузера.

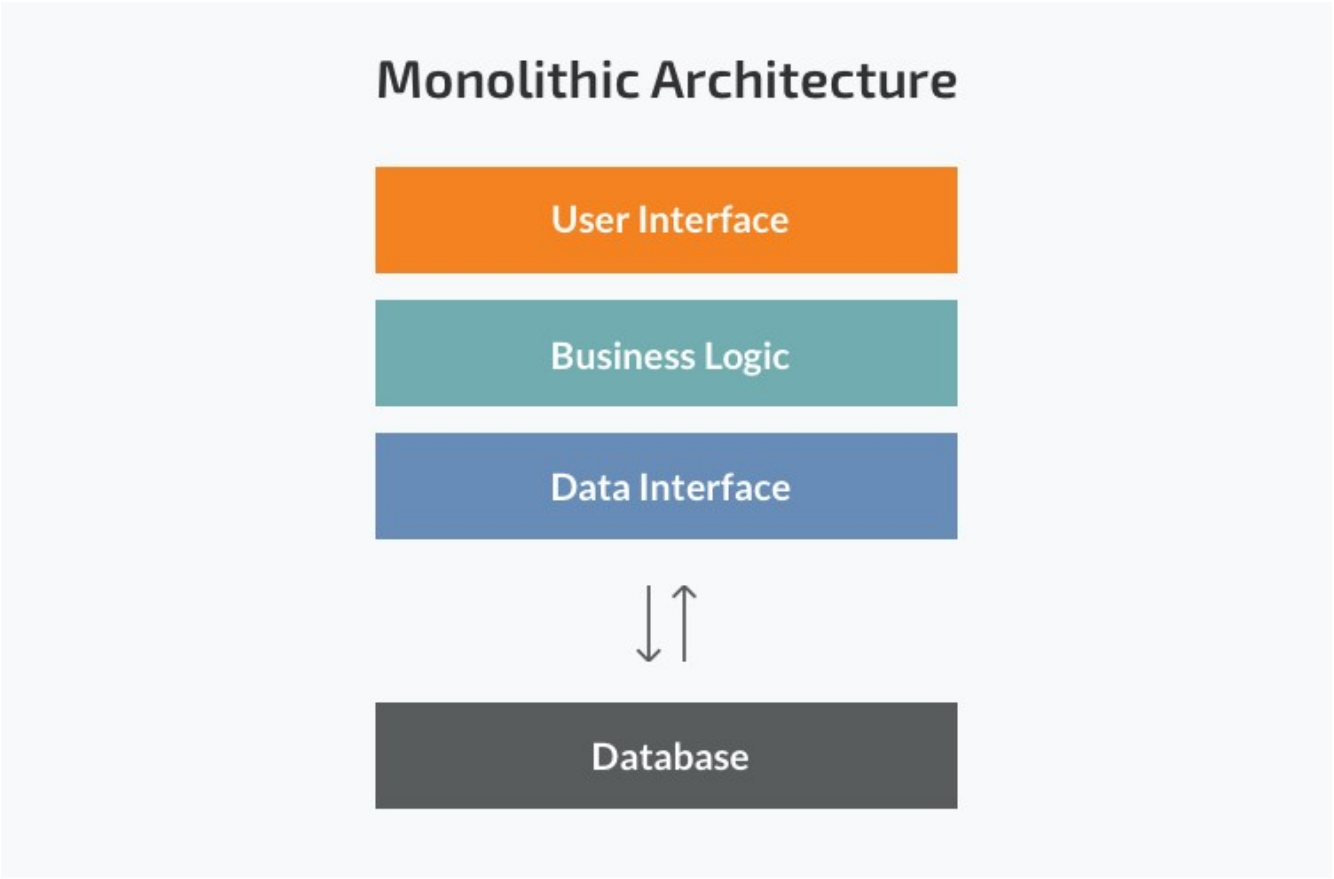


Рисунок 1.4 – Структура монолітної архітектури

Ідея цього розділення полягає в роботі з будь-якими компонентами архітектури, незалежно від того, що знаходиться нижче або вище.

Сильні сторони монолітної архітектури [6].

– Менше наскрізних проблем.

Наскрізні проблеми - це проблеми, які впливають на всю програму, такі як реєстрація, обробка, кешування та моніторинг продуктивності. У монолітній програмі ця область функціональності впливає лише на одну програму, тому нею легше керувати.

– Простіше налагодження та тестування.

На відміну від мікросервісної архітектури, монолітні програми набагато легше налагоджувати та тестувати. Оскільки монолітна програма є єдиною, неподільною одиницею, наскрізне тестування можна зробити набагато швидше.

– Просто поширювати.

Ще однією перевагою простоти монолітних додатків є простіший розподіл. Що стосується монолітних додатків, то вам не потрібно керувати багатьма дистрибутивами - лише одним файлом або одним каталогом.

– Проста в розробці на початку.

Оскільки монолітний підхід є стандартним способом створення додатків, будь-яка інженерна команда має знання та навички, необхідні для розробки монолітного додатку.

Слабкі сторони монолітної архітектури.

– Розуміння.

Коли монолітна програма зростає, вона стає надто складною для розуміння. Також важко керувати складною системою коду в одному додатку.

– Зміни.

Складніше внести зміни до такого великого і складного додатка з дуже щільними зв’язками. Всякий раз, коли вам потрібно створити, протестувати або розгорнути, ви по суті працюєте з усією базою коду. А на запуск будь-якої функції підуть дні. Одне розгортання може містити компоненти доступу до даних, компоненти бізнес-логіки, веб-служби тощо. Кожна зміна коду впливає на всю систему, тому вона повинна бути ретельно скоординована. Це значно затримує загальний процес розробки.

– Масштабованість.

Масштабувати компоненти не можна самостійно, лише всю програму. Кожного разу, коли ви збираєте, тестуєте та розгортаєте, вам потрібно модифікувати весь моноліт, оскільки модулі сильно залежать один від одного. Монолітна архітектура є найбільш ефективною для невеликих проєктів з чітко визначеною областю, де ви навряд чи зможете постійно підтримувати або розвивати свою базу коду. Додаток з одного джерела можна розгорнути у хмарі, і ви все ще можете використовувати ресурси зберігання.

– Супутні витрати на інфраструктуру.

Якщо один компонент завантажений і його потрібно масштабувати, вам потрібно буде додати ресурси для всього додатка. Це означає, що неефективна частина програмної архітектури може зруйнувати всю структуру, або вам доведеться заплатити багато грошей, щоб вона працювала.

– Бар’єри для нових технологій.

Незалежно від того, наскільки простими можуть здатися початкові кроки, монолітні програми намагаються застосувати нові та передові технології. Вкрай проблематично застосовувати нову технологію в монолітній програмі, оскільки всю програму доведеться переписати. Оскільки зміни мов або фреймворків впливають на весь додаток, потрібно докласти зусиль для ретельної роботи з деталями програми, а це означає, що це займає багато часу та зусиль.

1.2.2 Сервісно-орієнтована архітектура

Природним переходом від монолітного програмного додатку стало використання архітектури, орієнтованої на сервіси [6]. Сервісно-орієнтована архітектура – це архітектурний підхід для визначення, об'єднання та інтеграції чітко визначених та автономних багаторазових бізнес-послуг із власними можливостями [5]. Ці сервіси взаємодіють між собою, щоб забезпечити легку передачу даних, або передбачати координацію двох або більше сервісів. Складність кожного сервісу в сервісно-орієнтованому програмному додатку, як правило, дуже низька, і вони спілкуються між собою за допомогою низки API. За допомогою цього підходу програмний додаток ділиться на менші модулі. Отже, усі сервіси працюють на рівні агрегації, який можна назвати шиною [6]. Схема сервісно-орієнтованої архітектури зображена на рисунку 1.5.

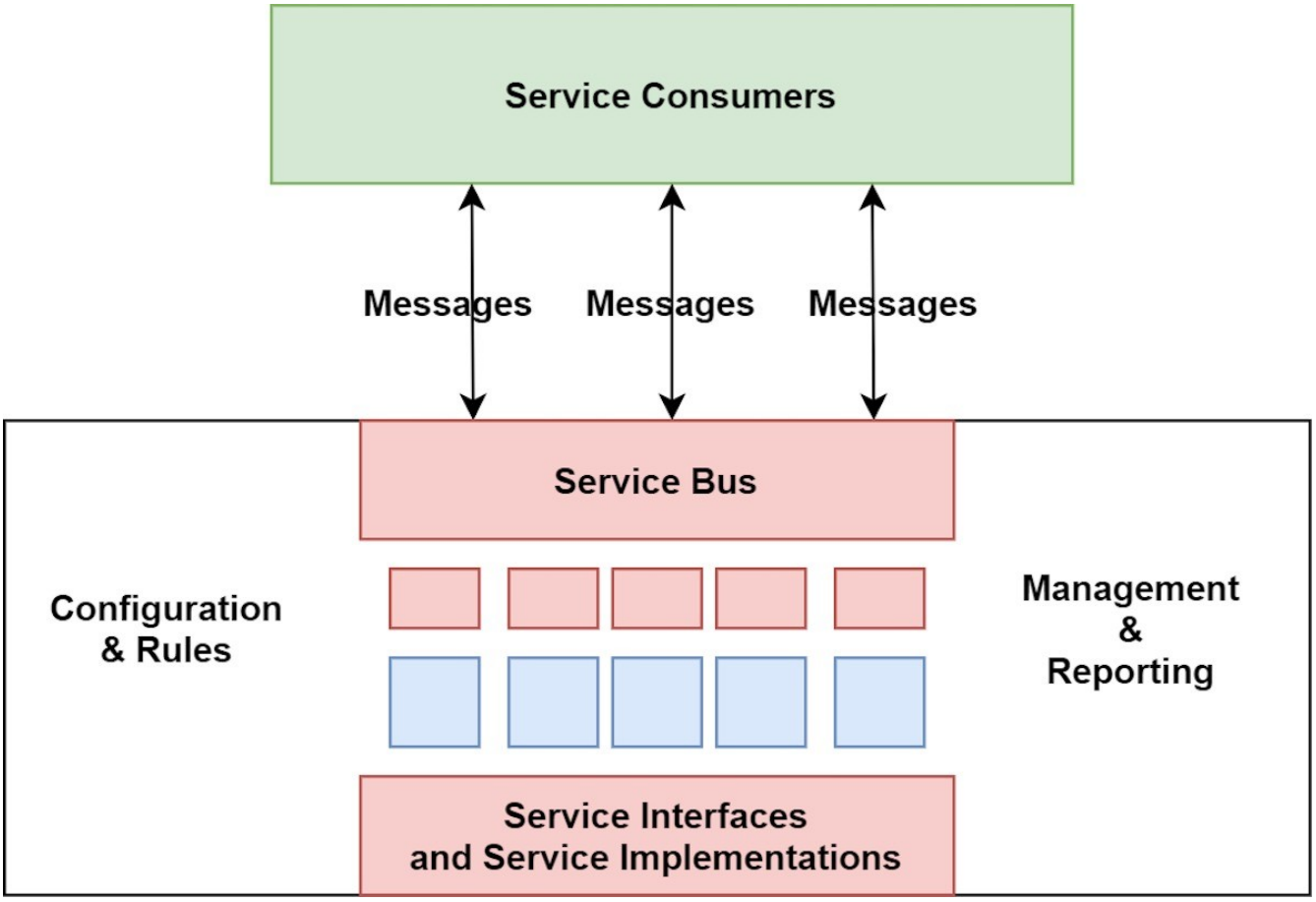


Рисунок 1.5 – Структура сервісно-орієнтованої архітектури

Прикладом, коли зазвичай використовується сервісно-орієнтована архітектура, є веб-сайт порівняння автострахування, який отримує доступ до баз даних та надає ділові дані та технічні деталі для побудови графічного інтерфейсу користувача.

Сильні сторони сервісно-орієнтованої архітектури [7].

– Гнучкість.

Сервісно-орієнтована архітектура пропонує велику гнучкість у створенні складних архітектур, і один компонент не порушить решту, якщо реалізація піде не так.

– Структурування.

Хороший спосіб роз'єднання та комунікації відповідає цій архітектурі. Кожен сервіс може взаємодіяти, використовуючи стандартний корпоративний рівень обробки. Цей шар розділяє внутрішні та зовнішні елементи системи.

– Легкий в обслуговуванні.

Змінити та оновити будь-який сервіс, реалізовану в сервісно-орієнтованій архітектурі, дуже просто. Не потрібно оновлювати всю систему. Сервіс управляється третьою стороною, і будь-які зміни в цьому сервісі не вплинуть на загальну систему. У більшості випадків старіша версія API працює, оскільки вона працювала раніше.

– Незалежність від платформи.

Сервіси спілкуються з альтернативними додатками спільною мовою, що означає, що програма не залежить від платформи. Сервіси можуть надавати API різними мовами, такими як PHP, JavaScript та інші.

– Масштабованість.

Якщо сервіс залучає велику кількість користувачів, його часто можна просто розширити, підключивши додаткові сервери. Таким чином, запити користувачів завжди будуть оброблятися.

Слабкі сторони сервісно-орієнтованої архітектури.

– Шина.

Цей рівень агрегації (шина сервісно-орієнтованої архітектури) є найбільшим викликом, з яким доводиться стикатися. Проблема полягає в додаванні операційної логіки до шини. Оскільки цей рівень стає все більшим і більшим, і все більше і більше компонентів додається до системи, виникають системні проблеми зв’язування.

– Обробка помилок.

Ще одна велика проблема пов’язана з обробкою помилок. За допомогою цієї архітектури ви отримаєте HTTP відповідь 200 або 500 від програмного забезпечення і нічого між ними. Тому виявити помилки авторизації або автентифікації, а також помилки, пов’язані з відсутністю деяких ресурсів, необхідних користувачеві системи, може бути важко.

– Складність отриманої архітектури.

Очевидним мінусом є те, що, хоча сервіси прості, архітектура може стати надто складною, щоб задовольнити зростаючі потреби бізнесу. Якщо окремі

функції загальної архітектури можуть не знадобитися, цього робити не слід, оскільки це тягне за собою великі витрати на людські та фінансові ресурси.

Розростання агрегації призвело до того, що ця сервісно-орієнтовна архітектура зникла з поля зору архітекторів. Люди почали звертати увагу на монолітні архітектури або перейшли на архітектуру мікросервісів [6].

1.2.3 Мікросервісна архітектура

Мікросервісна архітектура – це підхід до розробки додатків, де великий додаток будується як сукупність модульних сервісів (тобто слабо зв’язаних модулів / компонентів) [8]. Кожен модуль підтримує певну бізнес-мету і використовує простий і чітко визначений інтерфейс для взаємодії з іншими наборами сервісів.

Замість спільного використання однієї бази даних, як у монолітній програмі, кожен мікросервіс має свою базу даних (рисунок 1.6). Наявність бази даних для кожного мікросервіса важливо для отримання максимуму користі від мікросервісів, оскільки це забезпечує слабку зв’язність. Кожен із сервісів має свою базу даних. Крім того, мікросервіс може використовувати тип бази даних, який найкраще відповідає потребам.

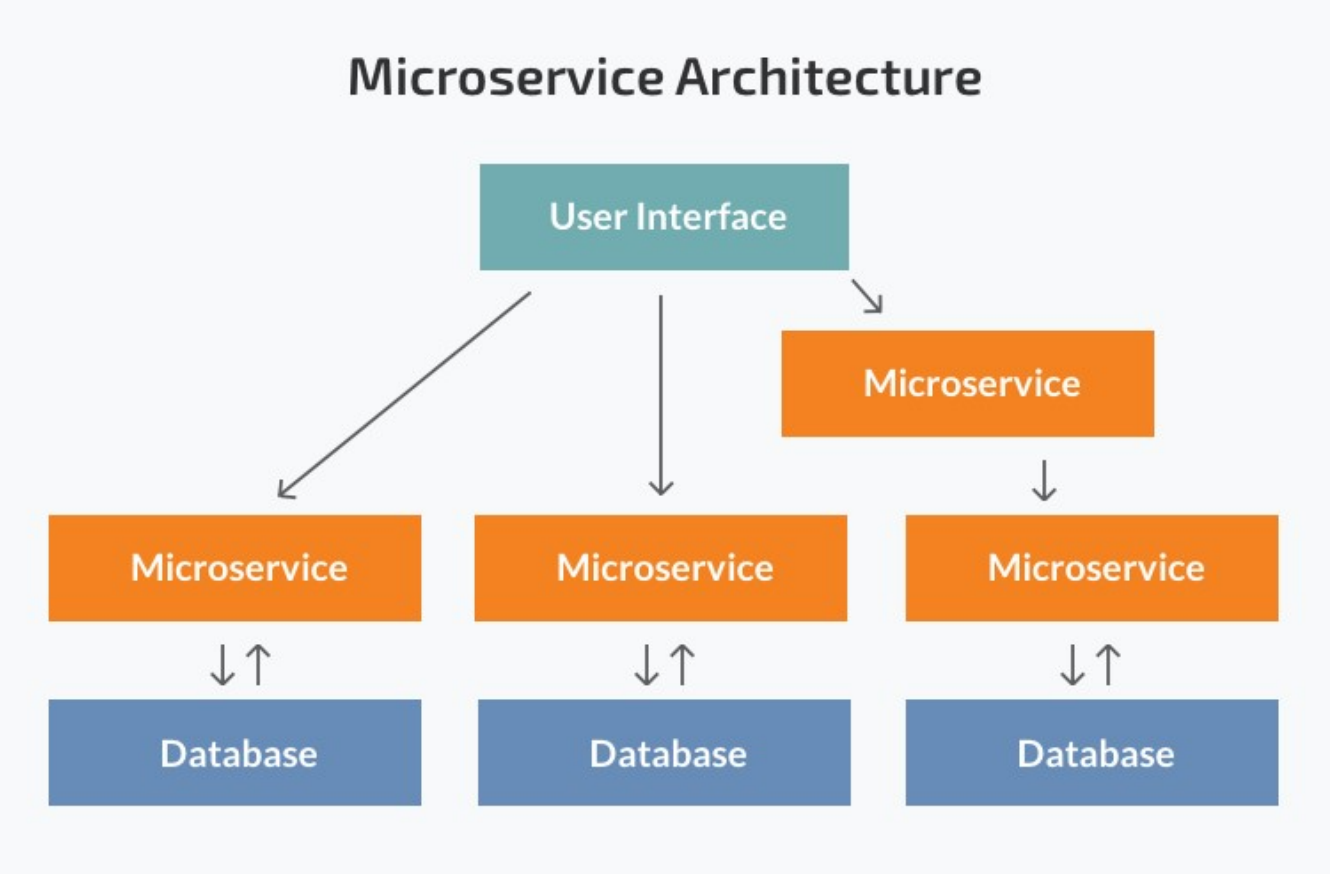


Рисунок 1.6 – Структура мікросервісної архітектури

Мистецтво та майстерність правильної побудови цієї архітектури походить від здатності визначати мікросервіс [9]. Занадто детальний або занадто широкий вигляд може зруйнувати архітектуру. Серед варіантів визначення меж мікросервісу ви можете використовувати бізнес-кейси, ієрархію або сегментацію доменів для визначення кожного мікросервісу.

Ця архітектура дозволила реалізувати справжній поліглот, в якому можуть використовуватися різні мови або платформи для співпраці. Отже, розробники інтерфейсу можуть працювати з React, тоді як команда, що працює над серверами, використовує C#, а команда по роботі з даними - Python. Усі мікросервіси можуть спілкуватися між собою та за потреби використовувати ресурси.

Весь зв’язок між мікросервісами відбувається через REST через HTTP. Один з найкращих способів налаштувати весь зв’язок для мікросервісів - це через проксі-API.

Ця архітектура також підходить для хмарних розгортань. При правильному виконанні ця архітектура дійсно допомагає у створенні хмарних сервісів. Більшість архітекторів працюють з мікросервісами, щоб перейти до хмарної платформи.

По суті, мікросеріси - це еволюція SOA. Ця архітектура дозволила розділити або розкласти систему на окремі робочі одиниці. Порівнюючи мікросервіси з SOA, вони обидва покладаються на сервіси як на їх основний компонент, але вони сильно відрізняються з точки зору характеристик сервісів [10].

– Поняття сервісу.

Сервісні компоненти в архітектурі мікросервісів, як правило, є сервісами єдиноцільового використання, які дійсно роблять одне. З SOA розмір компонентів сервісу може варіюватися від невеликих додатків до великих бізнес-сервісів. Насправді SOA зазвичай має компонент сервісу, представлений великим продуктом або навіть підсистемою.

– Спільний доступ до компонентів.

Спільне використання компонентів є одним із основних принципів SOA. Дійсно, спільне використання компонентів - це те, що є сервісами. SOA покращує спільний доступ до компонентів, тоді як архітектура мікросервісів прагне мінімізувати спільний доступ через "обмежений контекст". Розмежований контекст означає підключення компонента та його даних як єдиного блоку з мінімальними залежностями. Оскільки SOA покладається на кілька сервісів для задоволення попиту бізнесу, системи на базі SOA, швидше за все, будуть повільнішими, ніж архітектури мікросервісів.

– Проміжне програмне забезпечення та рівень API.

Модель архітектури мікросервісів зазвичай має так званий рівень API, тоді як SOA має компонент проміжного програмного забезпечення для обміну повідомленнями. SOA проміжне програмне забезпечення надає безліч додаткових функцій, відсутніх в архітектурі мікросервісів, включаючи

посередництво та маршрутизацію, вдосконалення повідомлень та трансформацію повідомлень та протоколів. Мікросервісна архітектура має рівень API між послугами та споживачами послуг.

– Віддалені сервіси.

Архітектури SOA покладаються на обмін повідомленнями (AMQP, MSMQ) та SOAP як основних протоколів віддаленого доступу. Більшість архітектур мікросервісів базуються на двох протоколах: REST та простих повідомленнях (JMS, MSMQ), а протокол, що використовується в архітектурах мікросервісів, загалом однорідний.

– Неоднорідна сумісність.

SOA сприяє розповсюдженню багатьох різнорідних протоколів за допомогою компонента проміжного програмного забезпечення для обміну повідомленнями. Архітектура мікросервісів намагається спростити модель архітектури за рахунок зменшення кількості варіантів інтеграції. Якщо вам потрібно інтегрувати кілька систем із використанням різних протоколів в неоднорідному середовищі, слід розглянути SOA. Якщо до всіх сервісів можна отримати доступ за одним протоколом віддаленого доступу, найкращим варіантом є мікросервісна архітектура.

Сильні сторони мікросервісної архітектури [6].

– Незалежні компоненти.

По-перше, всі мікросервіси можуть бути розгорнуті та оновлені самостійно, що забезпечує велику гнучкість. По-друге, збій у мікросервісі впливає лише на п певний сервіс, а не на всю програму. Також набагато простіше додати нову функціональність до програми мікросервісів, ніж до монолітної програми.

– Легше розуміння.

Додаток для мікросервісної архітектури, розбитий на менші та простіші компоненти, легше зрозуміти та керувати ним. Для розробки вам потрібно зосередитись лише на певному мікросервісі, пов’язаній з певною бізнес-метою.

– Краща масштабованість.

Ще однією перевагою підходу мікросервісів є те, що кожен елемент можна масштабувати незалежно. Отже, весь процес дешевший та ефективніший у часі, ніж використання монолітів, де весь додаток потрібно масштабувати, навіть якщо це не потрібно. Крім того, кожен моноліт має обмеження щодо масштабованості, тому чим більше користувачів набувається, тим більші проблеми з монолітом. Тому багато компаній в кінцевому підсумку перебудовують свою монолітну архітектуру. Наприклад, Currencycloud потрібно було перейти на мікросервіси через збільшення кількості транзакцій, оброблених їх платформою. Заснована в 2012 році, компанія пропонує глобальну платформу для платіжних послуг B2B. Спочатку їхня монолітна архітектура могла обробляти кількість транзакцій, які вони мали. Однак у міру зростання успіху компанії вони потребували більш ефективного рішення, яке в подальшому може бути розширено. Тож вони перейшли до мікросервісів.

– Гнучкість у виборі технології.

Інженерні команди не обмежуються технологією, обраною з самого початку. Вони можуть вільно застосовувати різні технології та платформи для кожного мікросервісу.

– Вищий рівень гнучкості.

Будь-який збій у застосунку мікросервісів впливає лише на певний сервіс, а не на все рішення. Тому всі модифікації та експерименти виконуються з меншим ризиком та меншою кількістю помилок.

Слабкі сторони мікросервісної архітектури.

– Додаткова складність.

Оскільки мікросервісна архітектура є розподіленою системою, необхідно вибрати та налаштувати з'єднання між усіма модулями та базами даних. Крім того, якщо ця програма включає незалежні сервіси, усі вони повинні бути розгорнуті незалежно.

– Розподіл системи.

Архітектура мікросервісів - це складна система з декількох модулів та баз даних, тому всіма підключеннями слід ретельно керувати.

– Поширені проблеми.

Створюючи додаток з мікросервісною архітектурою, слід вирішити низку наскрізних питань. Сюди входять зовнішня конфігурація, логування, отримання та перевірка показників системи, перевірка працездатності тощо.

– Тестування.

Безліч компонентів, що розгортаються самостійно, значно ускладнює тестування рішення на мікросервісній архітектурі.

Загальне порівняння підходів до архітектури наведено в таблиці 1.1 [5] та на рисунку 1.7.

Таблиця 1.1 – Порівняння підходів до архітектури програмного забезпечення

Характеристика	Монолітна архітектура	SOA	Мікросервісна архітектура
Розробка	Для розробки монолітного додатку використовують єдиний стек технологій (таких як JEE або .NET), що обмежує інструменти необхідні під конкретну задачу інструментами всього додатку	Перевикористання компонентів та стандартизація підходів облегшують розробку.	Розробка мікросервісів окремо дозволяє розробникам використовувати необхідні під конкретні задачі технології розробки для вирішення поставленого завдання.

Продовження таблиці 1.1

Дизайн	Монолітний додаток розростається до величезних розмірів, коли вже стає майже неможливо зрозуміти роботу системи.	Сервіси можуть бути різного розміру, від маленького додатку до великого корпоративного рішення, яке має в собі значну частину бізнес-логіки.	Система поділена на маленькі формалізовані частини з чітко визначеним бізнес-інтерфейсом.
Використання	Обмежене перевикористання компонентів в системі.	Сервіси доступні за стандартним протоколом, таким як SOAP, та проміжним програмним забезпеченням другим сервісам для використання.	Сервіси доступні за стандартним протоколом, таким як REST, другим сервісам для використання.
Гнучкість	Складно досягти необхідної гнучкості при регулярному розгортуванні програмного забезпечення.	Велика кількість залежностей, що виникають через полегшення доступу до компонентів та ускладнюють управління.	Маленькі частини системи, що можна незалежно розгортувати та вводити в роботу гарантують високий рівень гнучкості.

Продовження таблиці 1.1

Масштабованість	Масштабувати монолітне програмне забезпечення важко та дорого.	Можуть виникати складнощі з масштабуванням через залежності між сервісами та спільно використовувані модулі.	Оскільки сервіси - окремі модуль, що можна розгортати незалежно, масштабування дуже облегшується.

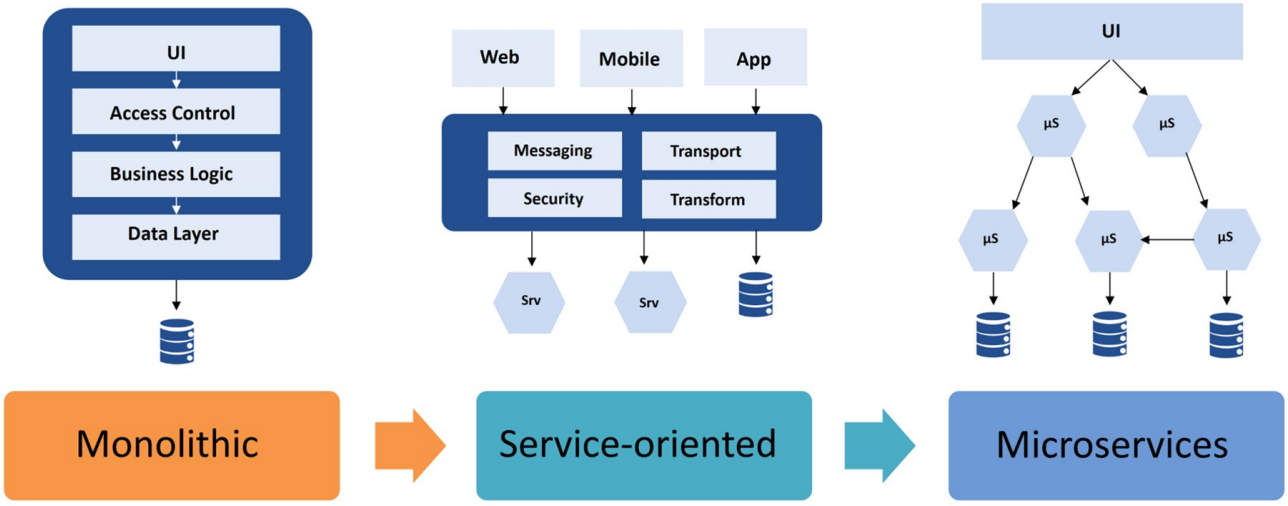


Рисунок 1.7 – Порівняння підходів до архітектури програмного забезпечення

1.3 Постановка завдання

Мета.

Основною метою даної роботи є підвищення ефективності паралельної реалізації алгоритму імітації Петрі-об’єктних моделей для надвеликих моделей.

Призначення.

Призначенням даної роботи є створення програмного забезпечення імітації дискретно-подійних систем, що зменшить час імітації та дозволить

імітувати надвеликі моделі за рахунок покращеного паралельного алгоритму імітації Петрі-об’єктних моделей.

Задачі.

Для досягнення мети роботи необхідно вирішити такі задачі:

- аналіз формалізмів дискретно-подійних систем;
- порівняння підходів до архітектури систем;
- дослідження паралельного алгоритму імітації Петрі-об’єктних моделей;
- розробка розподіленого алгоритму імітації Петрі-об’єктних моделей;
- моделювання та конструювання сервісу імітації.

1.4 Висновки до розділу

Дискретно-подійне моделювання - потужний інструмент для представлення, аналізу та оптимізації систем. Було розглянуто поняття дискретно-подійного моделювання та два його формалізми: систему масового обслуговування та мережі Петрі. Система масового обслуговування більш зрозуміла та прямолінійна, а мережі Петрі дозволяють набагато більш гнучко представляти моделі. Тому для сервісу паралельної імітації дискретно-подійних систем краще обрати саме мережі Петрі, оскільки вони покривають більше випадків, з якими може стикнутися користувач.

Крім того, було розглянуто три основні підходи до архітектури сервісів: монолітна архтітектура, сервісно-орієнтована архітектура та мікросервісна архітектура. З їх порівняння можна зробити висновок, що розробляємому сервісу краще підходить мікросервісна архітектура, адже вона дозволить ефективно управляти системою при збільшенні навантаження та сервіс.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Опис архітектури

Сервіс паралельної імітації дискретно-подійних систем потребує високого рівня масштабованості та гнучкості, оскільки може бути сильно навантажений, але не постійно. Тому потрібно, щоб система була спроєктована так, щоб була можливість додавати та забирати ресурси для окремих частин сервісу, наприклад, віддавати більше ресурсів для частини, що проводить імітацію, але не збільшувати кількість ресурсів, що відводиться під валідацію. Мікросервісна архітектура - рішення цієї проблеми.

Вже розроблена велика кількість шаблонів мікросервісної архітектури, які допомагають вирішувати найбільш поширені проблеми, з якими доводиться стикатися розробникам. Загальна схема цих шаблонів наведена на рисунку 2.1. Далі розглянуті шаблони, які знадобляться для проєктування архітектури сервісу паралельної імітації дискретно-подійних систем [11].

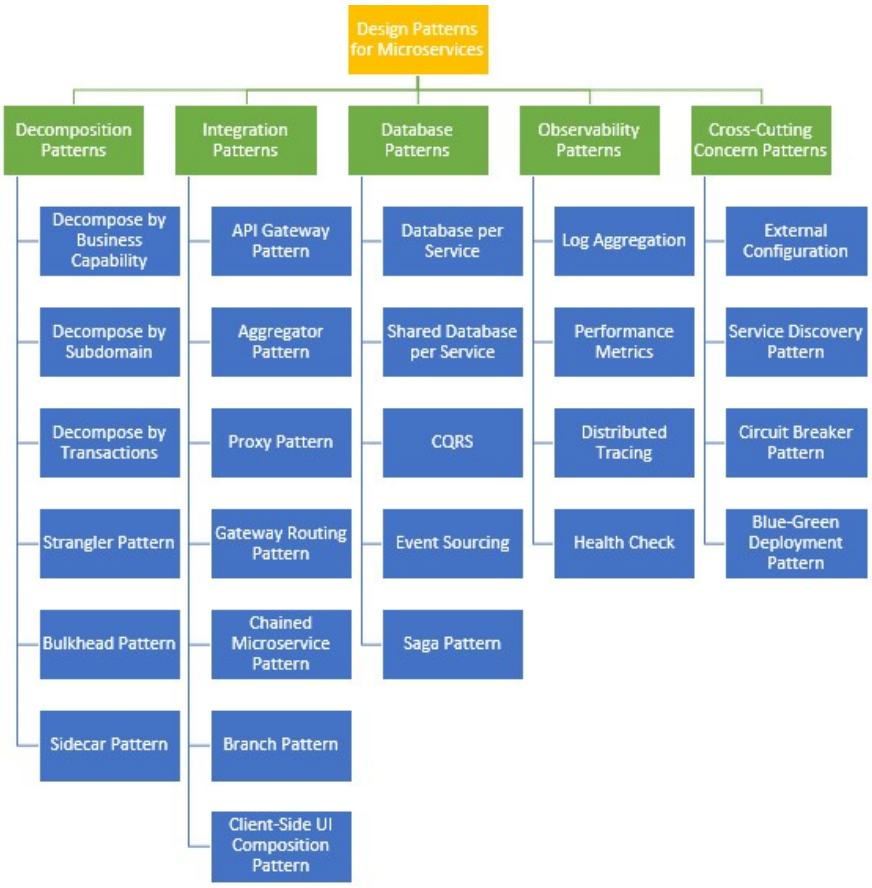


Рисунок 2.1 – Загальна схема шаблонів проєктування в мікросервісній архітектурі

2.1.1 Шаблон “Decompose by Business Capability”

Суть мікросервісів - це створення вільно пов'язаних між собою сервісів з застосуванням принципу єдиної відповідальності. У цій моделі мікросервіс визначається бізнес-можливостями. Для цього потрібно визначити послуги, які відповідають біснес-можливостям. Бізнес-можливість - це концепція моделювання бізнес-архітектури [12]. Це те, що компанія робить для створення вартості. Бізнес-можливість часто відповідає бізнес-об'єкту, наприклад:

- за замовлення відповідає "Управління замовленнями";
- за клієнтів відповідаю "Управління клієнтами".

2.1.2 Шаблон “Decompose by Subdomain”

Декомпозиція програми за допомогою шаблону бізнес-можливостей може бути хорошим початком, але розробники можуть зіткнутися з такими

проблемами, як "божественні класи", які буде нелегко розбити. Ці класи будуть спільними для кількох сервісів. У цьому випадку потрібно визначити сервіси, що відповідають субдоменам Domain-Driven Design (DDD). DDD відноситься до проблемного простору програми - компанії - як домену. Домен складається з декількох субдоменів. Кожен субдомен відповідає різній частині бізнесу.

Субдомени можна класифікувати наступним чином:

- ядро – це ключовий фактор бізнесу та найцінніша частина програми;

- підтримуючий субдомен: пов’язаний з тим, що робить компанія, але не є відмінною рисою. Вони можуть бути впроваджені як внутрішніми розробниками, так і зовнішніми;

- стандартний субдомен: неспецифічні частини бізнесу та в ідеальному випадку реалізовані із використанням стандартного програмного забезпечення;

Наприклад, субдомени управління замовленнями включають:

- сервіс каталогу товарів;

- сервіс управління запасами;

- сервіс управління замовленнями;

- сервіс управління доставкою.

2.1.3 Шаблон “API Gateway”

Коли програма розбита на менші мікросервіси, потрібно вирішити кілька проблем.

- проблема декількох викликів для декількох мікросервісів на різних каналах;

- потреба управління різними типами протоколів;

- різні споживачі можуть вимагати різні формати відповідей.

Шлюз API допомагає вирішити багато питань, порушених впровадженнями мікросервісів, але не обмежуючись переліченими вище:

- шлюз API – це єдина точка входу для виклику будь-якого мікросервісу;

- може виступати в ролі проксі-сервісу для перенаправлення запиту до відповідної мікросервісу;

- може узагальнювати результати, що надсилаються споживачеві;

- це рішення може створювати докладні API для кожного конкретного типу клієнта;

- він також може трансформувати запит і відповідь протоколу;

- шлюз API також може зняти відповідальність за автентифікацію / авторизацію з мікросервісу.

2.1.4 Шаблон “Aggregator”

Розбиваючи бізнес-функції на кілька менших логічних фрагментів коду, потрібно подумати, як взаємодіяти з даними, що повертаються кожним сервісом. Цю відповідальність не можна покласти на споживача.

Шаблон "Агрегатор" допомагає вирішити цю проблему. Розглянемо те, як можливо агрегувати дані з різних сервісів, а потім надіслати остаточну відповідь споживачеві. Це можна зробити двома способами [13]:

1. Складений мікросервіс викликає всі необхідні мікросервіси, консолідує дані та трансформує дані перед тим, як відправити їх назад;

2. Шлюз API також може розділити запит на декілька мікросервісів та агрегувати дані перед тим, як відправити їх споживачеві.

Якщо вам потрібно застосувати ділову логіку, рекомендовано вибрати складений мікросервіс. Якщо ні, API-шлюз є прийнятим рішенням.

2.1.5 Шаблон “Chained Microservice”

Наприклад, може бути декілька залежностей для окремих сервісів або мікросервісів: мікросервіс продаж має залежність від мікросервісу товарів та мікросервісу замовлень. Шаблон дизайну зв’язаних мікросервісів допомагає забезпечити єдиний результат для запиту. Запит, отриманий мікросервісом-1,

який потім взаємодіє з мікросервісом-2 і може взаємодіяти з мікросервісом-3. Всі ці послуги викликаються синхронно.

2.1.6 Шаблон “Branch”

Мікросервісу може знадобитися отримання даних із кількох джерел, включаючи інші мікросервіси. Шаблон мікросервісної архітектури "Branch" являє собою комбінацію шаблонів проєктування агрегатор та об'єднаних мікросервісів і дозволяє одночасно обробляти запити / відповіді від двох або більше мікросервісів. Микросервіс, що визивається, може бути ланцюжком мікросервісів. Шаблон Brach також може бути використаний для виклику різних ланцюжків мікросервісів або єдиного ланцюжка залежно від потреб бізнесу.

2.1.7 Шаблон “Event Sourcing”

Більшість програм працюють з даними, і зазвичай програма зберігає поточний стан. Наприклад, у традиційній моделі створення, читання, оновлення та видалення (CRUD) типовим процесом обробки даних є зчитування даних зі сховища. Він містить обмеження щодо блокування даних, коли транзакції використовуються часто.

Шаблон “Event Sourcing” [14] визначає підхід до управління операціями з даними, які керуються послідовністю подій, кожна з яких записується до сховища лише для додавання. Код програми надсилає ряд подій, які обов'язково описують кожну дію, виконану з даними, до сховища подій, де вони зберігаються. Кожна подія - це сукупність змін даних (наприклад, AddedItemToOrder).

Події зберігаються в архіві подій, який діє як система запису. Як правило, події, опубліковані сховищем подій, використовуються для підтримки матеріалізованих поглядів сутностей, оскільки вони модифікуються діями в додатку, а також для інтеграції із зовнішніми системами. Наприклад, система

може підтримувати матеріалізований вигляд усіх замовлень на продаж, що використовуються для заповнення частин інтерфейсу користувача. Коли програма додає нові замовлення, додає або видаляє товари до замовлення та додає інформацію про доставку, події, що описують ці зміни, можуть бути оброблені та використані для оновлення матеріалізованого подання. На рисунку 2.2 наведено огляд шаблона.

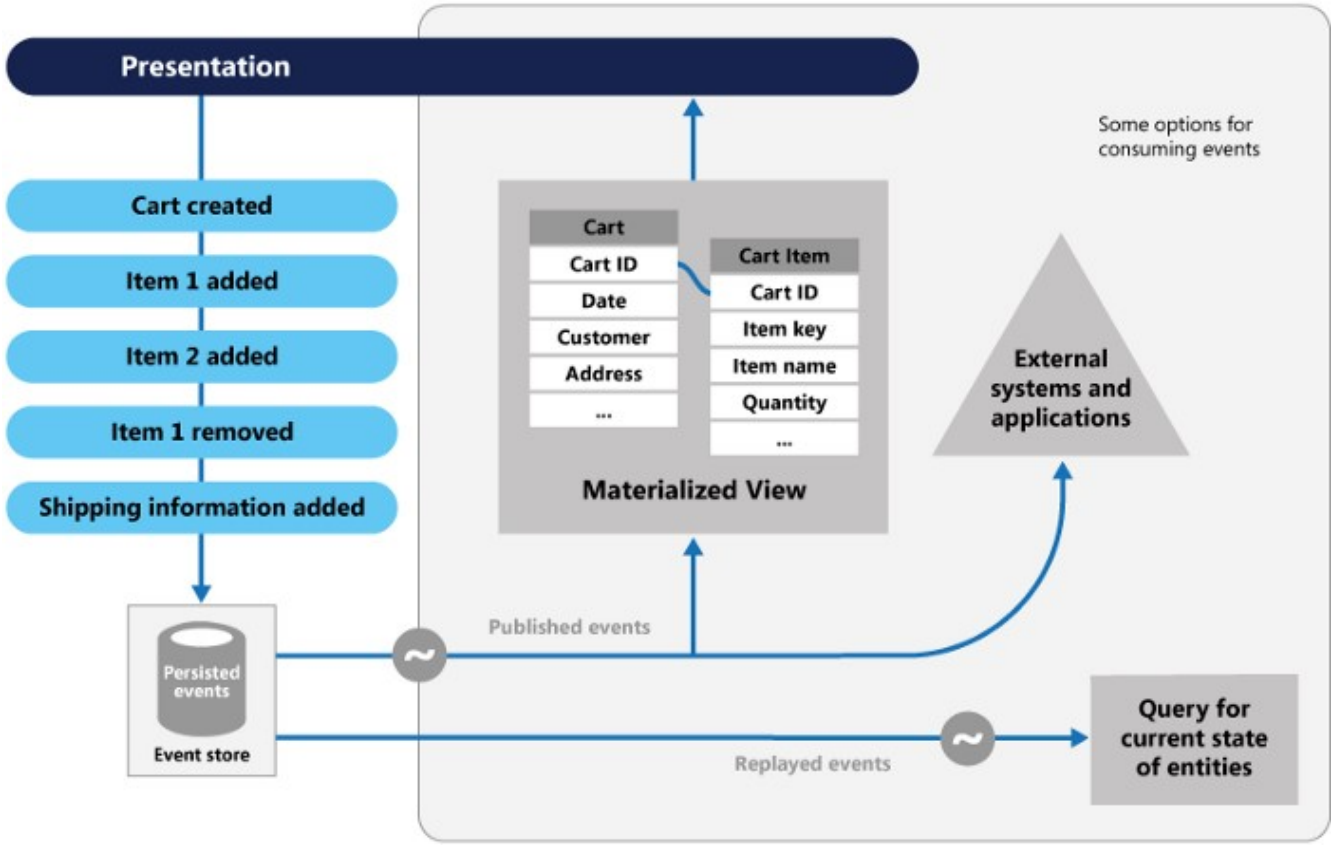


Рисунок 2.2 – Огляд шаблона “Event sourcing”

2.1.8 Шаблон “Performance Metrics”

У міру збільшення кількості сервісів в архітектурі мікросервісів стає надзвичайно важливим відстежувати транзакції, щоб можна було відстежувати патерни та надсилати сповіщення при виникненні проблеми.

Сервіс метрик потрібен для збору статистичних даних про окремі транзакції. Він повинен агрегувати метрики сервісів додатку, які надають звіти та попередження. Існує дві моделі агрегування метрик:

- push: служба надсилає метрики до служби метрик, наприклад NewRelic, AppDynamics;

- pull: служби метрик витягують метрики із такої служби, як, наприклад, Prometheus.

2.1.9 Шаблон “Health Check”

Після реалізації архітектури мікропослуг можливо, що сервіс працює, але він не може обробляти транзакції. Кожен сервіс повинен мати URL-адресу, за допомогою якої можна перевірити працездатність програми, наприклад / health. Цей API повинен перевіряти стан хосту, підключення до інших служб / інфраструктури та будь-яку конкретну логіку.

2.1.10 Шаблон “Service Discovery”

Коли мікросервіси вводяться в проєкт, необхідно вирішити кілька проблем, пов’язаних із викликами сервісів.

За допомогою контейнерної технології IP-адреси динамічно призначаються екземплярам сервісів. Щоразу, коли адреса змінюється, клієнтський сервіс може зупинятися та вимагати ручних змін.

Кожна URL-адреса сервісу повинна запам’ятовуватися споживачем і тісно пов’язана.

Потрібно створити список сервісів, який буде зберігати метадані для кожного сервісу та його специфікації. Екземпляр сервісу повинен зареєструватися в списку під час запуску та скасувати реєстрацію при вимкненні. Існує два типи сервісу виявлення:

- з боку клієнта: наприклад: Netflix Eureka;

- на стороні сервера: Наприклад: AWS ALB.

Схема моделі сервісу виявлення наведена на рисунку 2.3.

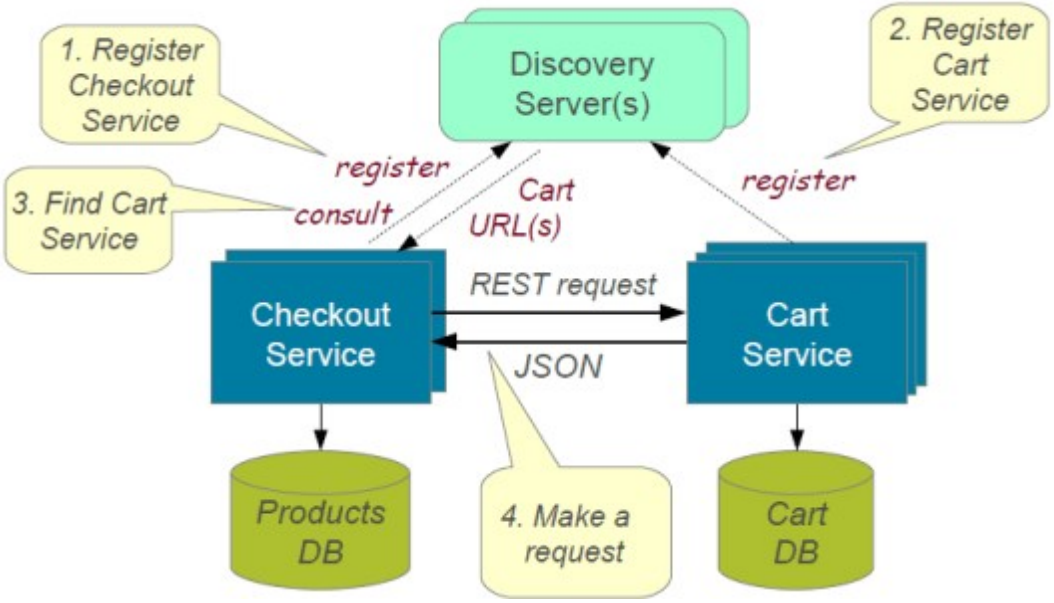


Рисунок 2.3 – Схема моделі сервісу виявлення

2.1.11 Аналіз алгоритму імітації

Для сервісу паралельної імітації дискретно-подійних систем була обрана імплементація дискретно-подійних систем - Петрі-об’єктні мережі. При прийнятті цього рішення було взято до уваги той факт, що мережі Петрі мають більш низький рівень абстракції ніж системи масового обслуговування, с чого слідує, що цей інструмент є більш універсальний. За рахунок цього сфера використання результуючого сервісу буде ширша.

У якості алгоритму, що буде використовуватися в сервісі було розглянуто паралельний алгоритм імітації Петрі-об’єктних моделей. В роботі [15] було його розроблено, основуючись на роботі [16], та досліджено, але дослідження проводилися на досить обмеженій кількості моделей. Не було проведено експериментів ні на достатньо великих моделях, ні на моделях різноманітної структури. Експерименти проводилися лише на моделі, зображеній на рисунку 2.4.

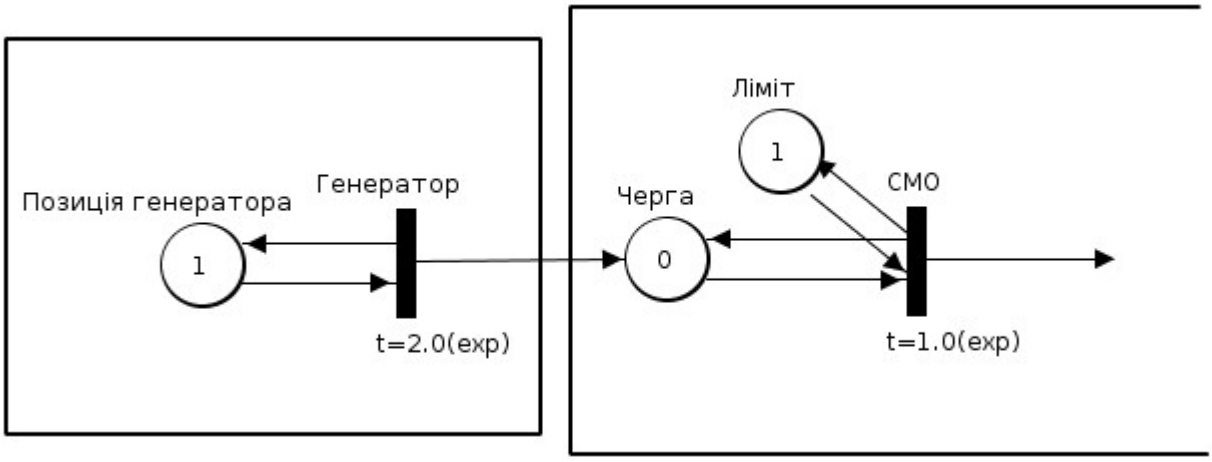


Рисунок 2.4 – Петрі-об’єктна модель, на якій проводилися експерименти

Спочатку Петрі-об’єкт генератора додавався до Петрі-об’єктної моделі, а до нього послідовно підключались від п’яти до п’ятдесяти Петрі-об’єктів, які в свою чергу формувалися з десяти СМО. Кожен перехід, що відображає СМО виконував вихід маркерів у чергу наступної СМО. Таким чином, загальна кількість переходів змінювалась від п’ятдесяти до лише п’ятисота з кроком п’ятдесят.

Щоб переконатися в доцільності вибору цього алгоритму необхідно провести ще ряд експериментів з різною структурою моделі та максимальною кількістю переходів хоча б тисяча.

В рамках цієї роботи було проведено експерименти на моделях трьох різних структур.

– Дерево, що розширюється.

Структура складається з Петрі-об’єкта генератора, який виходом під’єднаний до декількох (двох або п’яти) інших Петрі-об’єктів, кожен із яких в свою чергу також під’єднані до декількох Петрі-об’єктів і так далі, поки загальна кількість Петрі-об’єктів не досягне потрібної. Схематично моделі структури дерева, що розширюється, зображені на рисунку 2.5. Прямокутниками зображені Петрі-об’єкти внутрішня структура яких аналогічна структурі на рисунку 2.4.

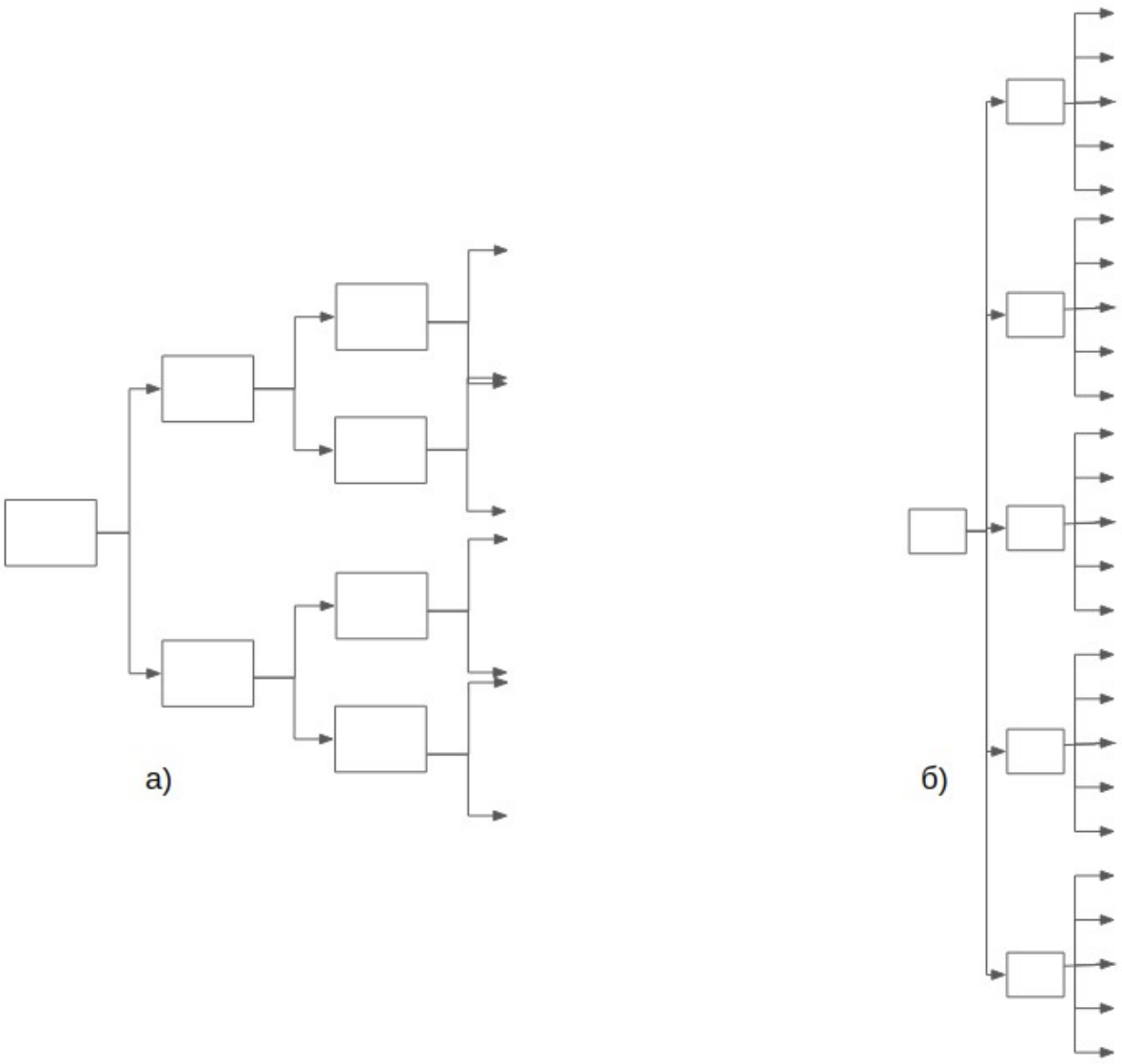


Рисунок 2.5 – Модель структури дерева, що розширюється (а) на дві гілки, (б) на п’ять гілок

– Дерево, що звужується.

Структура складається з Петрі-об’єкта генератора, який виходом під’єднаний до усіх листових Петрі-об’єктів дерева, після чого на кожному шарі дерева Петрі-об’єкт приєднаний входом до виходів декількох (двох або п’яти) інших Петрі-об’єктів і так далі, поки не буде досягнутий шар дерева з єдиним Петрі-об’єктом. Схематично моделі структури дерева, що звужується, зображені на рисунку 2.6. Прямокутниками зображені Петрі-об’єкти внутрішня структура яких аналогічна структурі на рисунку 2.4.

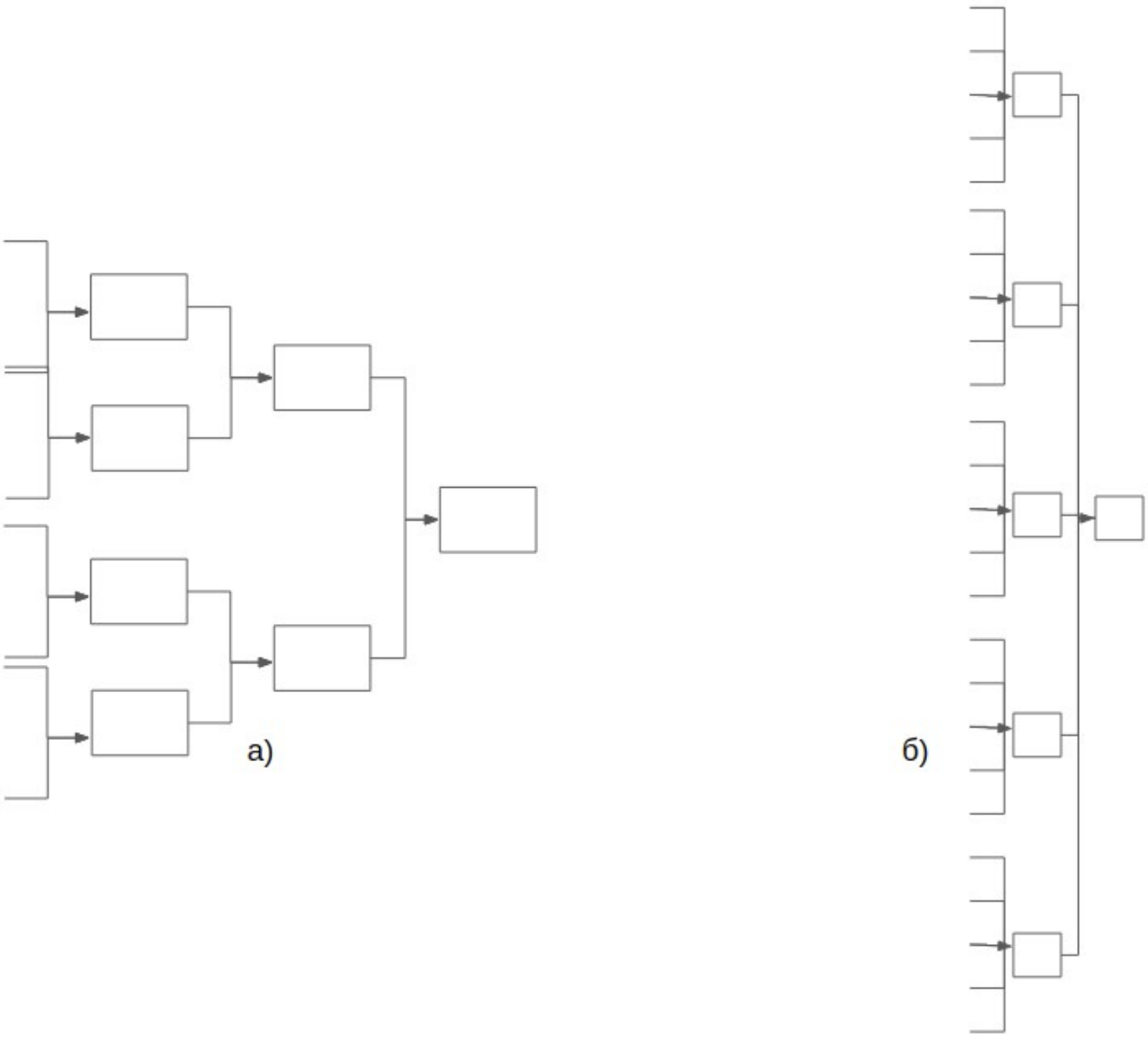


Рисунок 2.6 – Модель структури дерева, що звужується (а) у дві гілки, (б) у п’ять гілок

– Ланцюг, що розходиться та сходиться.

Структура складається з Петрі-об’єкта генератора, який виходом під’єднаний до декількох (двох або п’яти) інших Петрі-об’єктів, кожен із яких в свою чергу під’єднані до єдиного Петрі-об’єкта, а той знову до декількох і так далі поки загальна кількість Петрі-об’єктів не досягне потрібної. Схематично моделі структури дерева, що розширюється, зображені на рисунку 2.7. Прямокутниками зображені Петрі-об’єкти, внутрішня структура яких аналогічна структурі на рисунку 2.4.

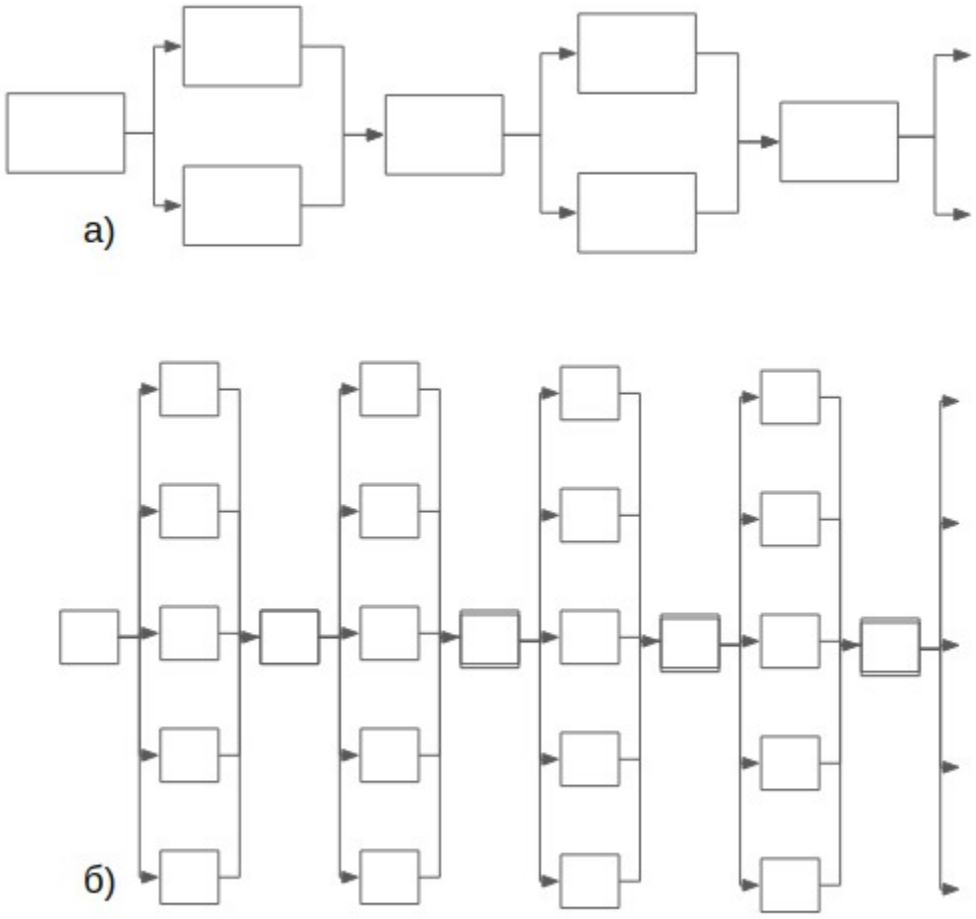


Рисунок 2.7 – Модель структури ланцюга, що розходиться та сходиться (а) у дві гілки, (б) у п’ять гілок

З Петрі-об’єктними моделями таких структур було проведені експерименти, в яких моделі були сформовані з від десяти до сотні Петрі-об’єктів, кожен з яких мав в собі по десять переходів. Таким чином загальна кількість переходів становила від сотні до тисячі. Замірювався реальний час роботи алгоритму та порівнювався з реальним часом роботи послідовного алгоритму. Експерименти проводилися на комп’ютері, характеристики якого наведені в таблиці 2.1.

Таблиця 2.1 – Характеристики комп’ютера, на якому проводилися експерименти

Характеристика	Значення
Процесор	Intel® Core™ i7-8650U CPU @ 1.90GHz × 8
Оперативна пам’ять	16 Гб
Графічний процесор	Mesa Intel® UHD Graphics 620 (KBL GT2)
Операційна система	Ubuntu 20.04.1 LTS

Результати експериментів для моделей дерев, що розширюються, для варіацією на дві гілки наведено на рисунку 2.8, а на п’ять гілок на рисунку 2.9.

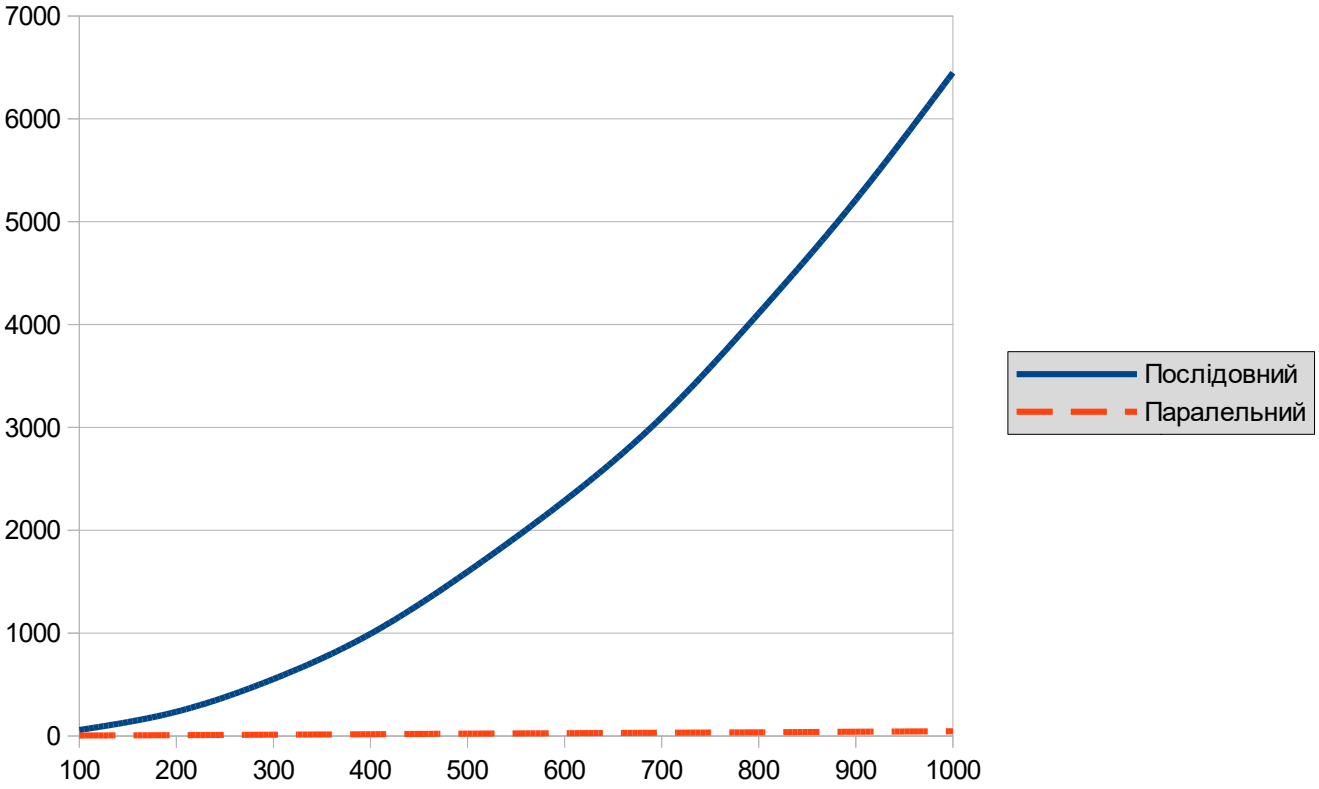


Рисунок 2.8 – Результати експериментів для дерева, що розширюється на дві гілки

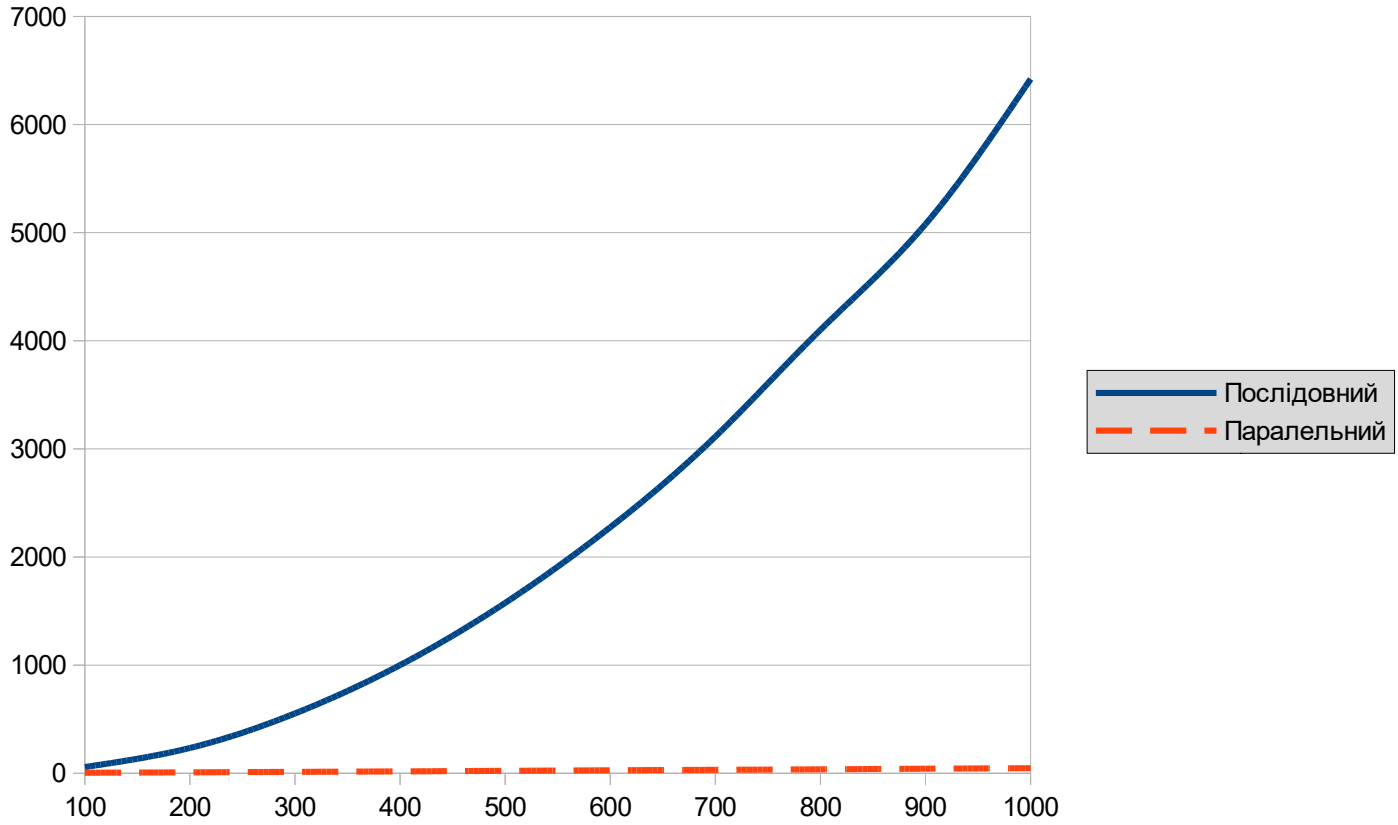


Рисунок 2.9 – Результати експериментів для дерева, що розширюється на п’ять гілок

З графіків видно, що різниця між часом роботи паралельного алгоритму незначна для моделей з різною кількістю гілок. Це краще видно на графіку порівняння швидкодії паралельного алгоритму (рисунок 2.10).

Однак більш цікавим може виявитися графік порівняння прискорення паралельного алгоритму відносно послідовного, наведений на рисунку 2.11. Зазвичай розпаралелювання алгоритмів не дає прискорення більше ніж в два рази, але у випадку паралельного алгоритму імітації Петрі-об’єктних моделей прискорення збільшується з розміром моделі. Наприклад, на моделі дерева, що розходиться на дві гілки, прискорення досягає 140 разів при розмірі моделі в тисячу переходів. Зростання прискорення при збільшенні моделі спостерігається також і для дерева, що сходиться (рисунок 2.12), і для ланцюга, що розходиться та сходиться (рисунок 2.13).

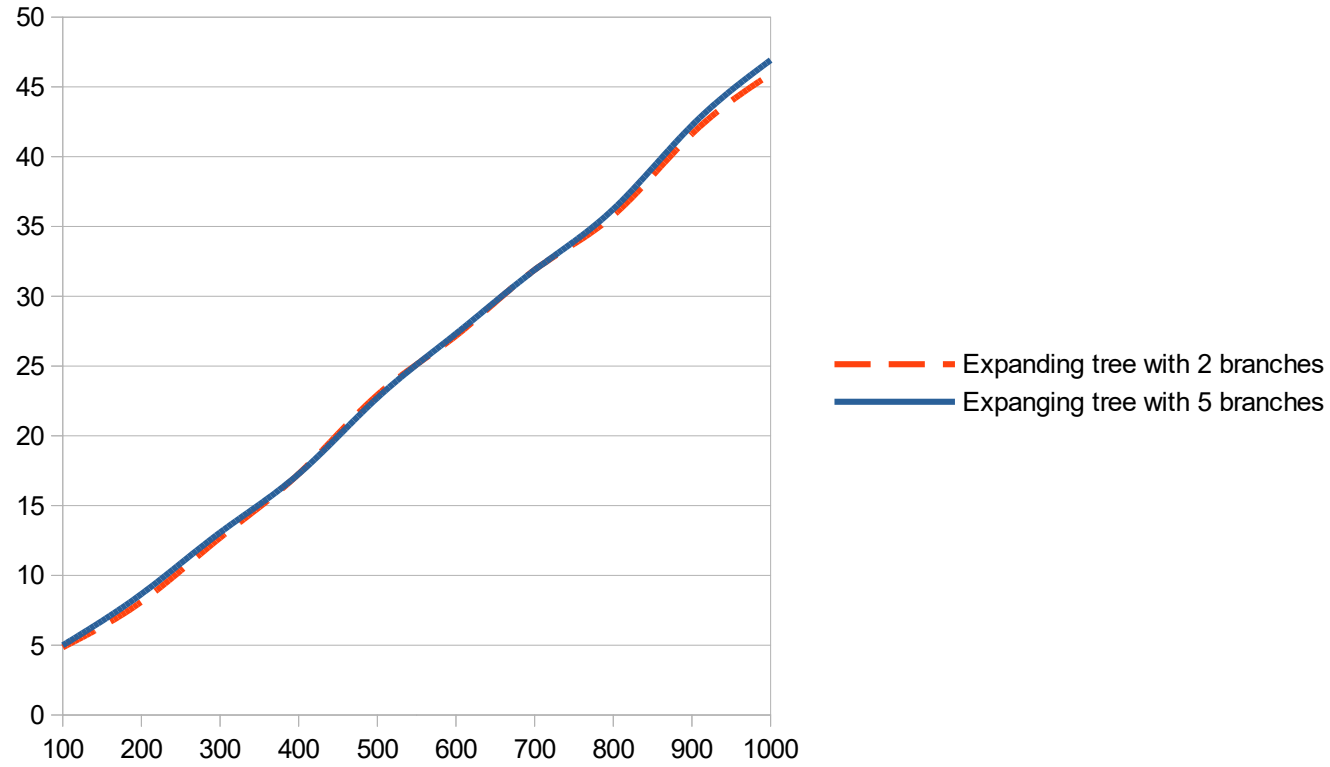


Рисунок 2.10 – Порівняння швидкодії паралельного алгоритму для дерев, що розширюється на дві та на п’ять гілок

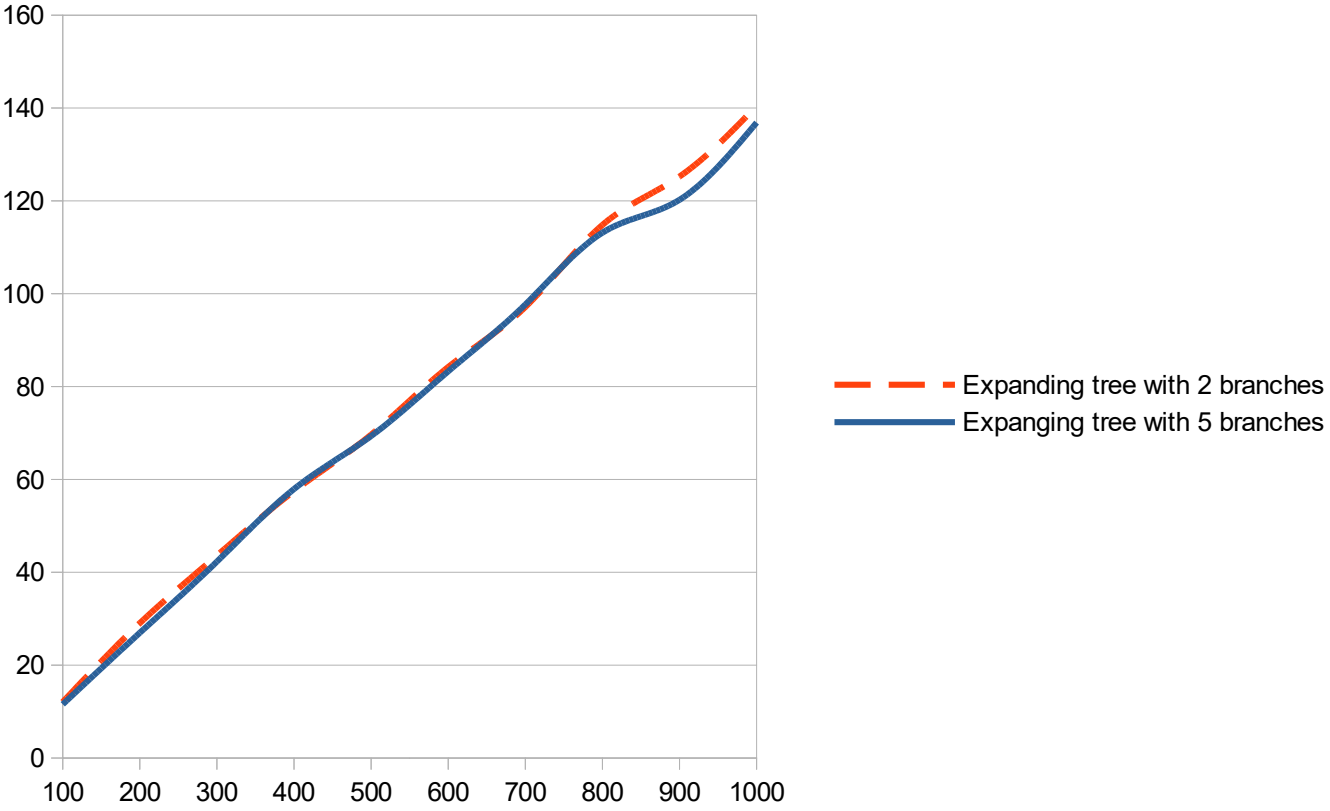


Рисунок 2.11 – Порівняння прискорення паралельного алгоритму відносно послідовного для дерев, що розширюється на дві та на п’ять гілок

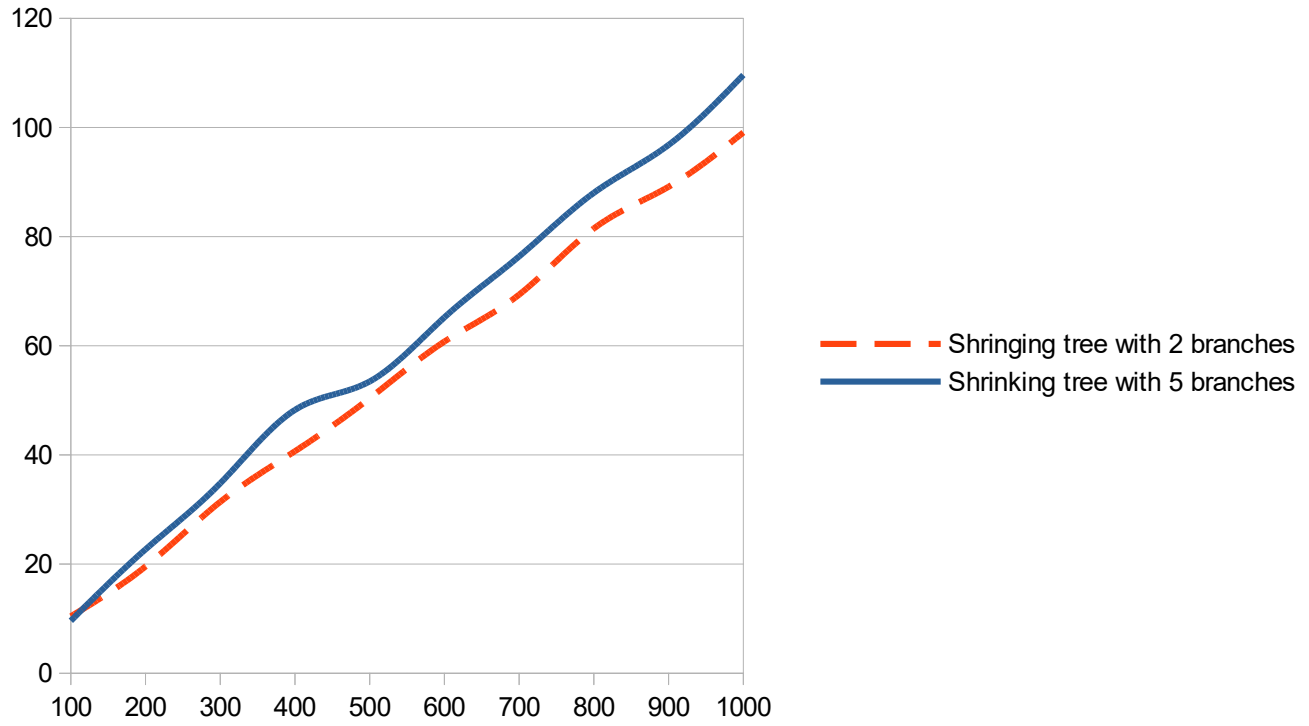


Рисунок 2.12 – Порівняння прискорення паралельного алгоритму відносно послідовного для дерев, що звужуються у дві та у п’ять гілок

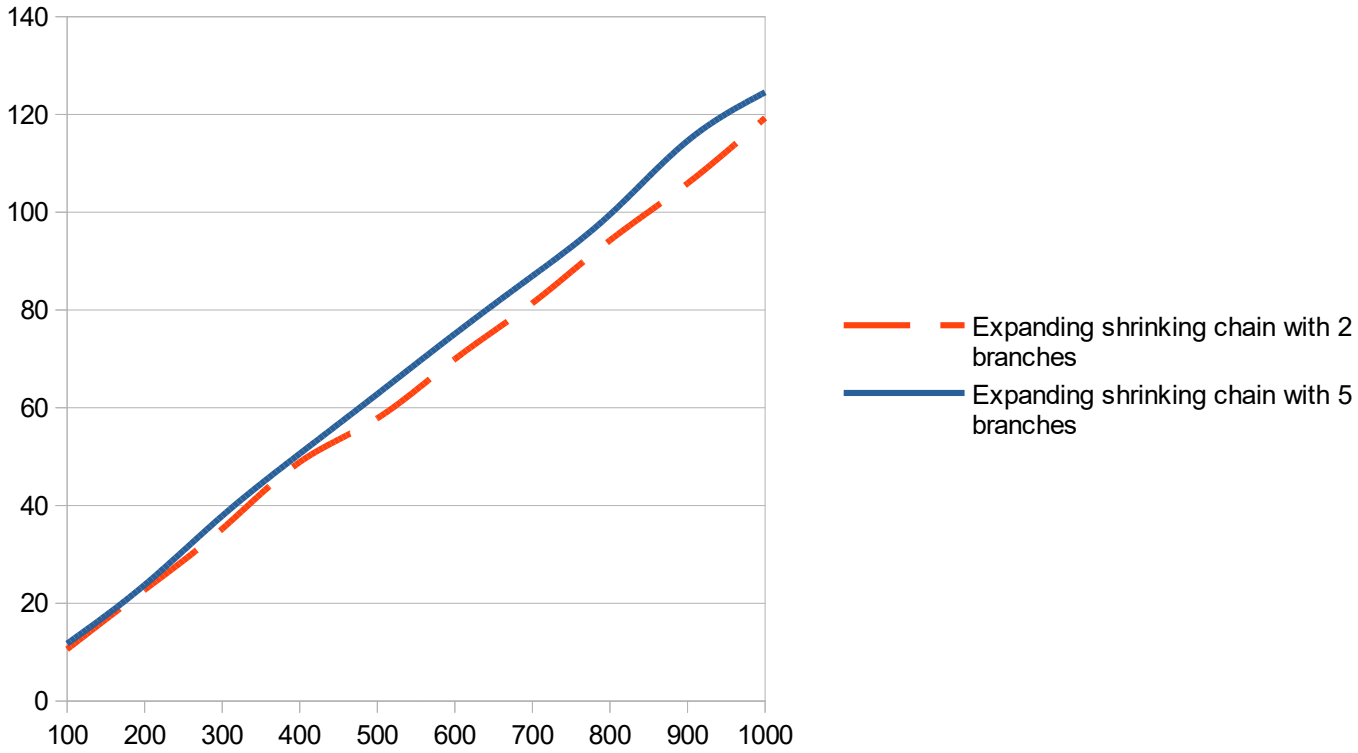


Рисунок 2.13 – Порівняння прискорення паралельного алгоритму відносно послідовного для данцюгів, що розширюються та звужуються у дві та у п’ять гілок

Графіки порівняння прискорення виглядають дуже схоже, але якщо звернути увагу на ось часу, то помітно, що прискорення доволі сильно відрізняється для різних моделей. Це проілюстровано на рисунку 2.14 для моделей на дві гілки та на рисунку 2.15 для моделей на п’ять гілок.

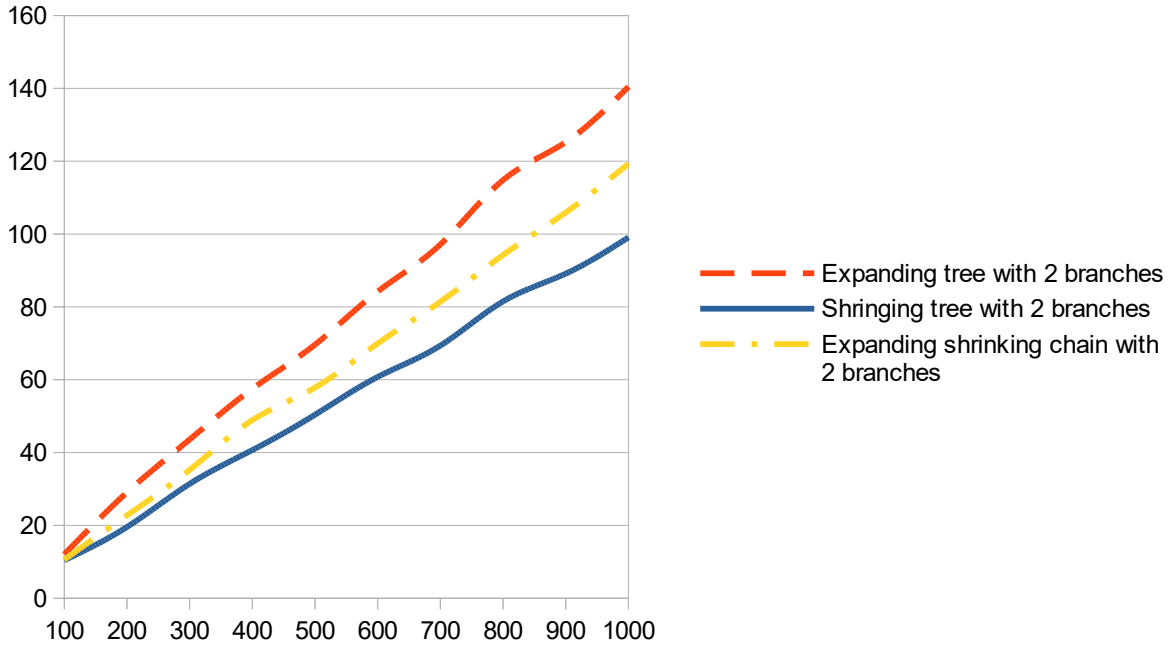


Рисунок 2.14 – Порівняння прискорення для всіх трьох типів моделей на дві гілки

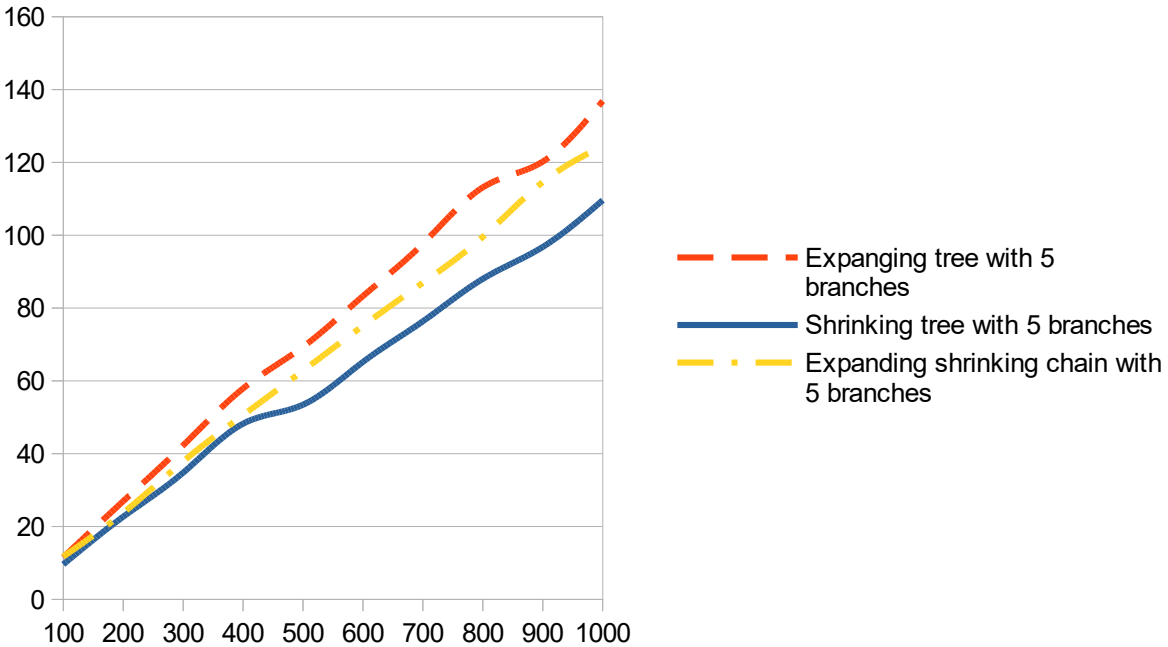


Рисунок 2.15 – Порівняння прискорення для всіх трьох типів моделей на п’ять гілки

Результат очікуваний – найбільше прискорення для моделі дерева, що розходиться, найменше - для дерева, що сходиться. Це пояснюється тим, що для першого варіанта моделі характерна зменшена кількість залежностей. Тобто, кожен Петрі-об’єкт активізує роботу декількох інших, і паралельний алгоритм може швидше просувати внутрішній час залежних Петрі-об’єктів. Для дерева, що звужується, навпаки, кожен об’єкт залежить від декількох інших і зможе просувати свій час тільки тоді, коли отримає події з усіх Петрі-об’єктів, що знаходяться до нього у моделі. Ланцюг, що розходиться та сходиться - це щось середнє між першими двома варіантами, тому і прискорення середнє.

Поясняється таке величезне прискорення тим, що паралельний алгоритм обчислення Петрі-об’єктних моделей - це не просто паралельна версія послідовного алгоритму, розглянутого в роботі [17], а зовсім інший алгоритм. Він просуває час імітації не для усієї системи, а для кожного Петрі-об’єкта фактично запускається послідовний алгоритм. Таким чином найгірший можливий випадок, з яким може стикнутися алгоритм (не враховуючи випадки, коли неможливо розбити модель на окремі компоненти сильної зв’язності) наступний: кожен Петрі-об’єкт передаватиме подію наступному тільки тоді, коли закінчиться його внутрішній час імітації. Тоді паралельний алгоритм буде просто послідовно запускати імітацію в кожному об’єкті і фактично не буде виконувати ніяких дій паралельно. Але це не те саме, що запустити послідовний алгоритм, оскільки його складність поліноміально залежить від кількості переходів в моделі. Тому виконати імітацію послідовно одну за одній на великій кількості частин моделі в багато разів швидше, ніж виконати імітацію цілої моделі. В реальних випадках до цього додається ще й те, що ці частини моделі здебільшого будуть імітуватися паралельно. Таким чином поліноміальною залежністю складності послідовного алгоритму від кількості переходів пояснюється залежність прискорення паралельного алгоритму відносно послідовного від розміру моделі.

Так як ми переконались в оптимальності паралельного алгоритму обчислення Петрі-об’єктних моделей, було прийнято рішення використовувати саме його в реалізації сервісу паралельної імітації дискретно-подійних систем.

2.2 Висновки до розділу

Другий розділ розглядає шаблони мікросервісної архітектури, які можуть бути використані при створенні сервісу паралельної імітації дискретно-подійних систем. За рахунок цих шаблонів можна уникнути розповсюджених проблем та задач, з якими стикаються розробники, що використовують мікросервісну архітектуру, та підвищити розуміння системи у цілому.

Також розглянуто та досліджено алгоритм, який можна взяти за основу при розробці сервісу. Проведено ряд експериментів, що доводять його ефективність для різноманітних систем та аналітично пояснена причина його пришвидшення у порівнянні з послідовним алгоритмом.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Загальна структура сервісу паралельної імітації дискретно-подійних систем зображено на рисунку 3.1. Складові сервісу: веб-клієнт, основний сервіс, сервіс повідомлень та вузлові сервіси, на яких проводиться власне паралельна імітація.

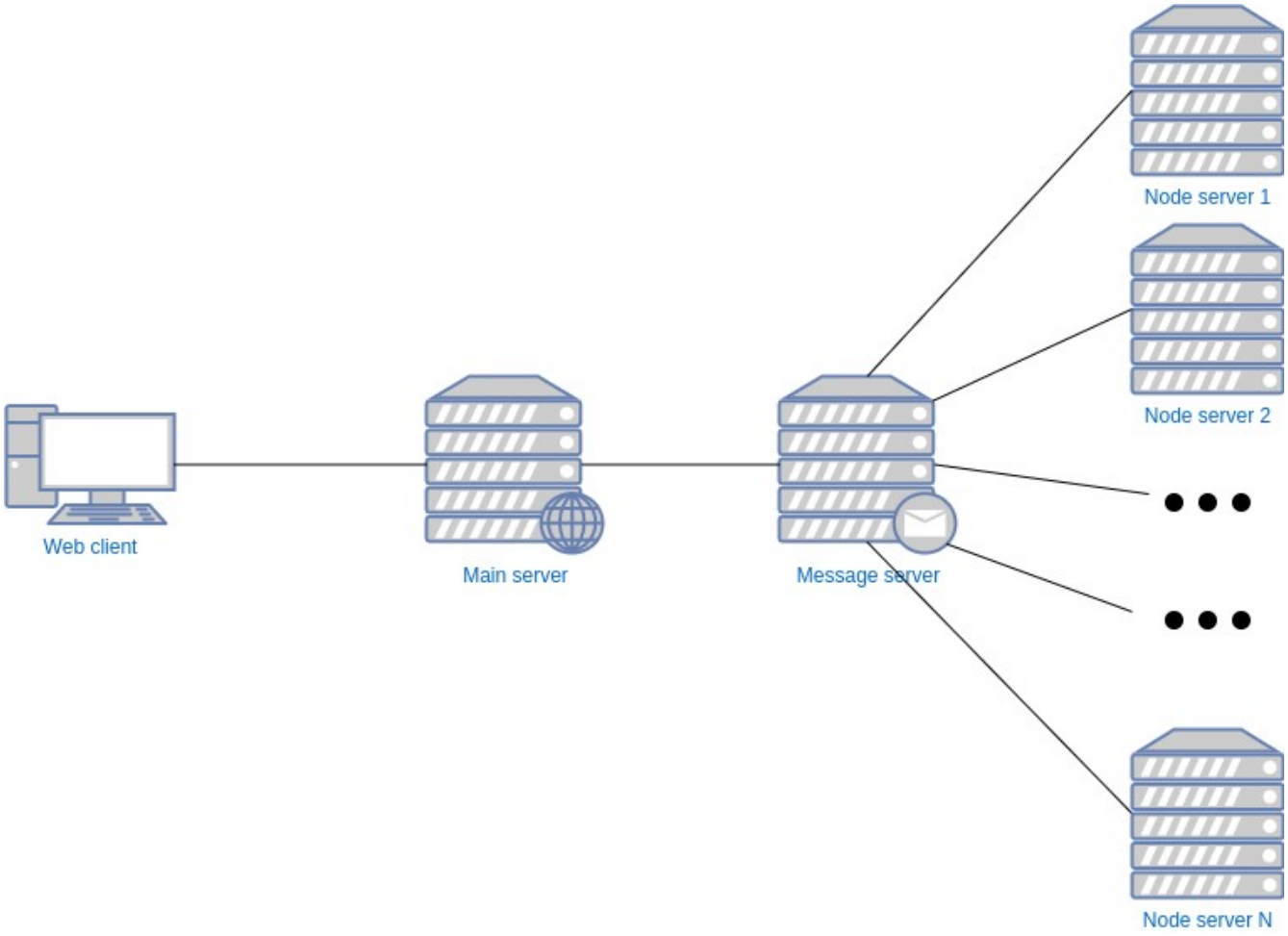


Рисунок 3.1 – Структура системи

3.2 Опис функцій частин системи

3.2.1 Вузлові сервіси.

Вузлові сервіси - ядро всієї системи. На них проводиться імітація дискретно-подійних систем. Саме в них використовується паралельний

алгоритм, розглянутий в попередньому розділі. Від їх доступності та правильності роботи залежать центральні функції усієї системи. Тому для розробки вузлового сервісу було обрано мову програмування Java, яка характеризується високим рівнем надійності.

Вузлові сервіси отримують через сервіс повідомлень від основного сервісу вже провалідовану та приведену до такої, що не має компонентів сильної зв'язності, частину Петрі-об'єктної моделі, яку потім подає на вхід алгоритму. За рахунок цього оптимізується робота вузлового сервісу - не потрібно витрачати ресурси на валідацію моделі та на перевірку, чи можливо імітувати її паралельним алгоритмом.

Паралельний алгоритм в вузлових сервісах оптимізовано для надвеликих моделей за рахунок того, що частини однієї моделі можуть обраховуватися на різних серверах. Для цього було додано нові реалізацію буферу передачі часу подій.

Буфер передачі часу подій використовувався в паралельному алгоритмі для того, щоб мати можливість незалежно просувати час в різних частинах моделі. Такий буфер в алгоритмі знаходиться між кожними залежними Петрі-об'єктами. Одні Петрі-об'єкти додавали час до буферу і продовжували свою роботу, коли інші очікували нових подій в буфері, щоб визначити інтервал часу, в якому вони можуть безпечно працювати.

Взаємодію між Петрі-об'єктами можна реалізувати через сервіс повідомлені, розподіливши його таким чином між різними серверами. Для цього попередню реалізацію буфера треба обернути в проксі (за шаблоном проєктування "Proxy") та додати логіку повідомлень.

У випадку, якщо буфер знаходиться на стороні, що отримує повідомлення, треба зареєструвати канал та додати колбек, метод, який буде викликаний при кожній вхідній події в буфер. На рисунку 3.2 наведено код реєстрації колбеку.

```
if (!sender) {
    try {
        ConnectionFactory factory = new ConnectionFactory();

        Connection connection = factory.newConnection();

        channel = connection.createChannel();

        channel.queueDeclare(arcId, b: false, b1: false, b2: false, map: null);

        DeliverCallback deliverCallback = (consumerTag, delivery) -> {
            receiveTime(DoubleToByteArrayUtil.toDouble(delivery.getBody()));
        };

        channel.basicConsume(arcId, b: true, deliverCallback, consumerTag -> {
        });
```

Рисунок 3.2 – Реєстрація колбеку

Зареєстрований метод має додавати час події в обернений буфер, що спонукає потік, що очікує на буфері, продовжити свою роботу. Крім того метод повинен закрити канал, з якого він викликається, коли отримує останнє повідомлення. Реалізація методу наведено на рисунку 3.3.

```
public void receiveTime(double time) {
    parallelEventTimesBuffer.addTime(time);
    if (channel != null && time == Double.MAX_VALUE) {
        try {
            channel.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Рисунок 3.3 – Метод, зареєстрований як колбек

На стороні серверу, в якому події додаються в буфер, алгоритм повинен додавати події в канал, який слухає буфер, що отримує. Для цього в проксі необхідно підмінити реалізацію методу addTime, як це наведено на рисунку 3.4.

```
public void addTime(double time) {
    ConnectionFactory factory = new ConnectionFactory();
    try (Connection connection = factory.newConnection();
         Channel channelOut = connection.createChannel()) {

        channelOut.basicPublish( s: "", arcId, basicProperties: null, DoubleToByteArrayUtil.toByteArray(time));
```

Рисунок 3.4 – Передача подій в канал з проксі

3.2.2 Сервіс повідомлень

Для сервісу повідомлень була обрана готове рішення RabbitMQ. RabbitMQ – це програмне забезпечення для чергування повідомлень, також відоме як брокер повідомлень або менеджер черг [18]. Коротко кажучи, – це програмне забезпечення, яке визначає черги, до яких підключаються програми для передачі одного або декількох повідомлень.

Повідомлення може містити будь-яку інформацію. Наприклад, воно може містити інформацію про процес або завдання, які потрібно виконати в іншій програмі (яка також може бути на іншому сервері), або це може бути просте текстове повідомлення. Програмне забезпечення диспетчера черг зберігає повідомлення, доки програма, що отримує, не з’єднається та не видалить повідомлення з черги. Потім програма-одержувач обробляє повідомлення.

Повідомлення не потрапляють безпосередньо в чергу. Натомість джерело надсилає повідомлення до модулю обміну. Модуль обміну відповідає за маршрутизацію повідомлень до різних черг за допомогою прив'язок та ключів маршрутизації. Прив’язка - це взаємозв'язок між чергою та модулем обміну.

Шлях повідомлення в RabbitMQ:

а) Джерело публікує повідомлення в модуль обміну. При створенні модулю обміну необхідно вказати тип (прямий, тема або інші);

б) Модуль обміну отримує повідомлення і тепер відповідає за маршрутизацію. Модуль обміну враховує різні атрибути повідомлення, наприклад ключ маршрутизації, залежно від типу модуля обміну;

в) Асоціації повинні створюватися модулем обміном до черг. У цьому випадку є два двосторонніх прив’язки, крім модулю обміну. Обмін направляє повідомлення в черги на основі атрибутів повідомлення;

г) Повідомлення залишаються в черзі, поки споживач їх не обробить;

д) Споживач обробляє повідомлення.

3.2.3 Основний сервіс

Основний сервіс системи виконує обробку клієнта та передоброку Петрі-об’єктної моделі. Оскільки імітація може займати багато часу, обробки простих HTTP запитів недостатньо для коректної роботи, так як на це дається лише шістдесят секунд. Рішенням цієї проблеми буде використання технології WebSocket.

З роботи [19] WebSocket — це комунікаційний протокол, що забезпечує повнодуплексні канали зв'язку по одному TCP-з'єднанню.

WebSocket відрізняється від HTTP. Обидва протоколу розташовані на рівні 7 в моделі OSI і залежать від TCP на рівні 4. Незважаючи на те, що вони різні, WebSocket призначений для роботи через порти HTTP 80 і 443, а також для підтримки проксі-серверів і посередників HTTP, таким чином він сумісний з протоколом HTTP.

Протокол WebSocket дозволяє взаємодіяти між веб-браузером (або іншим клієнтським додатком) і веб-сервером з меншими витратами, ніж напівдуплексні альтернативи полегшуючи передачу даних в режимі реального часу з сервера і на сервер. Це стало можливим завдяки наданню стандартизованим способом відправки сервером клієнтові контенту без попереднього запиту з боку клієнта і вирішення передачі повідомлень назад і вперед при збереженні відкритого з'єднання. Таким чином, між клієнтом і сервером може відбуватися двосторонній безперервний обмін повідомленнями. Зв'язок здійснюється через TCP-порт 80 (або 443 в разі з'єднань, зашифрованих по протоколу TLS), що корисно для тих середовищ, які блокують не інтернет з'єднання з використанням брандмауера.

Щоб отримувати повідомлення з WebSocket в сервісі було використано елементи фреймворку Spring. С початку необхідно налаштувати доступ до WebSocket. Приклад такої конфігурації наведено на рисунку 3.5.

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker( ...destinationPrefixes: "/topic");
        config.setApplicationDestinationPrefixes("/petri");
    }

    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint( ...strings: "/websocket").withSockJS();
    }

    public boolean configureMessageConverters(List<MessageConverter> messageConverters) {

        messageConverters.clear();

        MappingJackson2MessageConverter messageConverter = new MappingJackson2MessageConverter();
        messageConverter.getObjectMapper()
                .enable(DeserializationFeature.ACCEPT_EMPTY_STRING_AS_NULL_OBJECT);
        messageConverters.add(messageConverter);
        return false;
    }
}
```

Рисунок 3.5 – Конфігурація WebSocket в основному сервісі

Тепер можна отримувати та відправляти дані в WebSocket. Для цього потрібно створити Spring контроллер з вказанням шляхів з'єдняння, з якого сервіс отримує повідомленя та у який відправляє. На рисунку 3.6 наведено приклад такої вказівки.

```
@MessageMapping("/compute/distributed")
@SendToUser("/topic/statistics")
public PetriObjectModelStatisticsDTO computePetriObjectModelDistributed(
    @Valid @RequestBody PetriObjectModelDTO petriObjectModelDTO) {
```

Рисунок 3.6 – Налаштування Spring контроллеру для роботи з WebSocket

Отриману з WebSocket Петрі-об'єктну модель основний сервіс валідує та за допомогою алгоритму Косараджу приводить до такого, що не має компонентів сильної зв'язності. Після цього частини отриманої моделі основний сервіс надсилає до вузлових сервісом за алгоритмом round-robin.

Алгоритм round-robin розглянуто у роботі [20]. Round-robin (від англ. Round-robin - циклічний) - алгоритм розподілу навантаження розподіленої

обчислювальної системи методом перебору і впорядкування її елементів по круговому циклу.

Нехай є N об'єктів, здатних виконати задану дію, і M завдань, які повинні бути виконані цими об'єктами. Об'єкти n рівні за своїми властивостями між собою, завдання m мають рівний пріоритет. Тоді перше завдання (m = 1) призначається для виконання першого об'єкту (n = 1), друга - другого і т. д., До досягнення останнього об'єкта (m = N). Тоді наступне завдання (m = N + 1) буде призначена знову першому об'єкту і т. П. Простіше кажучи, відбувається перебір виконують завдання об'єктів по циклу, або по колу (round), і після досягнення останнього об'єкта наступне завдання буде також призначена першому об'єкту. Таким чином, алгоритм round-robin стає алгоритмом розподілу часу або балансування навантаження.

3.2.4 Веб-клієнт

Веб-клієнт надає користувачу можливість завантажувати Петрі-об’єктну модель у систему. Модель завантажується у вигляді JSON, що надає високу гнучкість, та дає можливість створювати будь-які моделі.

Крім того веб-клієнт повинен підтримувати WebSocket, відправляти модель в нього та отримувати відповідь сервера також з нього. Для цього при завантаженні сторінки JavaScript код створює SocketJS та підписується на події з нього, як показано на рисунку 3.7.

```
var stompClient = null;

connect();

function connect() {
    var socket = new SockJS('/websocket');
    stompClient = Stomp.over(socket);
    stompClient.connect({}, function (frame) {
        console.log('Connected: ' + frame);
        stompClient.subscribe( options: '/user/topic/statistics', function (statistics) {
            statsJson = JSON.parse(statistics.body);
            if (statsJson.message !== undefined) {
                showStatistics(statsJson.message);
            } else {
                showStatistics(JSON.stringify(JSON.parse(statistics.body), replacer: null, space: '\t'));
            }
        });
    });
}
```

Рисунок 3.7 – Налаштування WebSocket за допомогою SocketJS

Після цього повідомлення з серверу будуть оброблятися в методі showStatistics. Коли веб-клієнт відправляє модель на сервер, від надсилає її через сокет на спеціальний шлях, як вказано на рисунку 3.8.

```
function sendNet() {
    stompClient.send("/petri/compute/distributed", {}, $("#net").val());
}
```

Рисунок 3.8 – Відправлення повідомлення на сервер через WebSocket

3.3 Приклад роботи з додатком

Щоб отримати доступ до додатку необхідно завантажити основну і єдину сторінку, яка містить поле вводу JSON Петрі-об’єктної моделі та поле, у яку буде виведено статистику після підрахунку (рисунок 3.9).

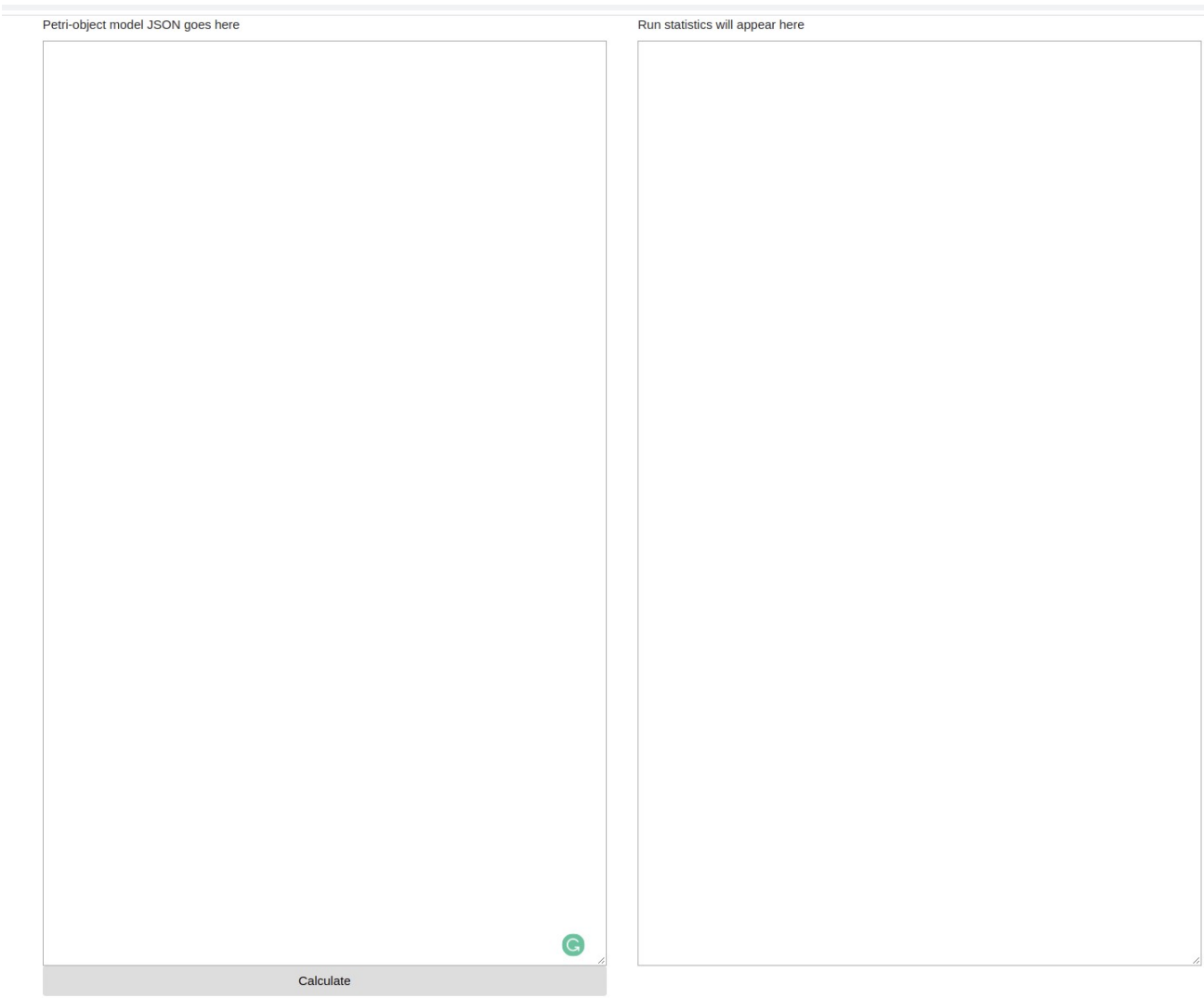


Рисунок 3.9 — Головна сторінка додатку

Сформований JSON Петрі-об’єктної моделі необхідно ввести в поле “Petri-object model JSON goes here” та натиснути кнопку “Calculate”. Приклад введення моделі наведено на рисунку 3.10, а вимоги до JSON Петрі-об’єктної моделі можна знайти в пункті “Формат вхідних та вихідних даних”.

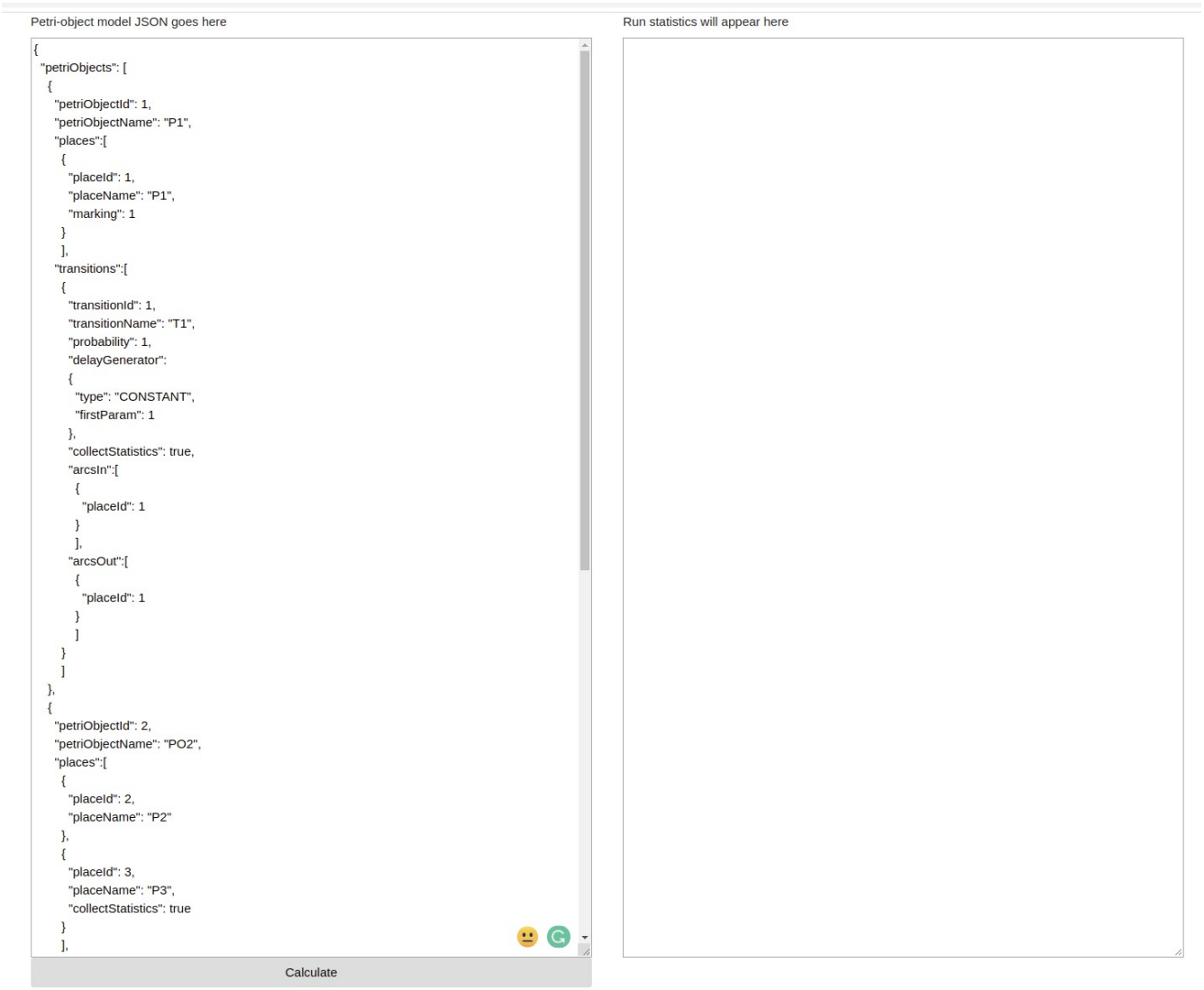


Рисунок 3.10 — Введення JSON Петрі-об’єктної моделі в додаток

Після натискання кнопки “Calculate” статистика обчислення моделі з’явиться в полі “Run statistics will appear here”, як тільки обчислення моделі виконається (рисунок 3.11).

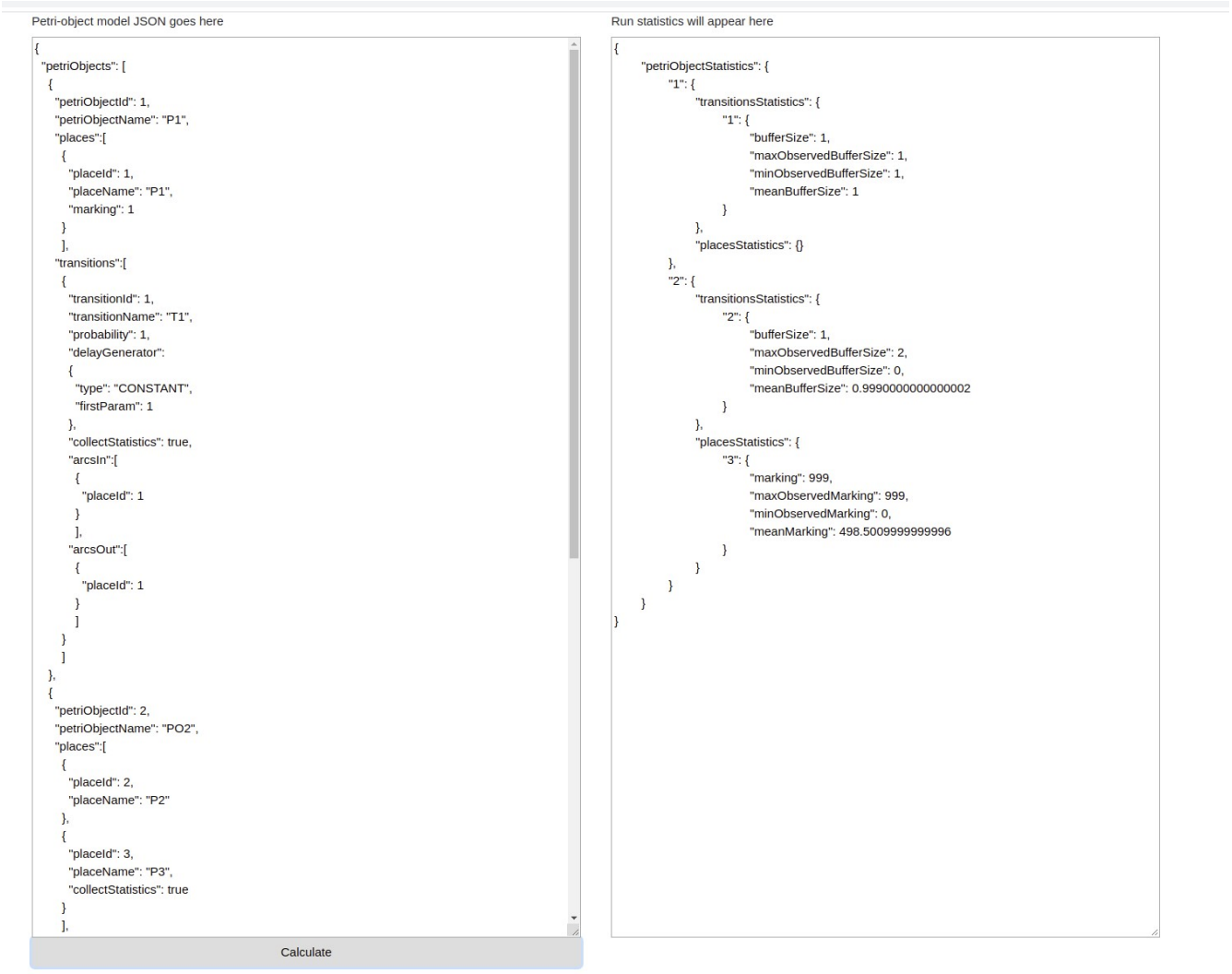


Рисунок 3.11 — Результат обчислення Петрі-об’єктної моделі

3.3.1 Формат вхідних та вихідних даних

На вхід системи подається JSON Петрі-об’єктної моделі. Параметри Петрі-об’єктної моделі наведено в таблиці 3.1

Таблиця 3.1 — Параметри Петрі-об’єктної моделі

Назва параметра	Тип параметра	Обов’язковий	Значення за замовчуванням	Опис параметра
petriObjects	Масив	Так	-	Список Петрі-об’єктів
externalArcs	Масив	Так	-	Список зовнішніх дуг
timeModelling	Додатне дробове число	Так	-	Час моделювання

Параметри Петрі-об’єкта наведено в таблиці 3.2.

Таблиця 3.2 — Параметри Петрі-об’єкта

Назва параметра	Тип параметра	Обов’язковий	Значення за замовчуванням	Опис параметра
petriObjectId	Ціле додатне число	Так	-	Ідентифікатор Петрі-об’єкта
petriObjectName	Строка	Так	-	Назва Петрі-об’єкта
places	Масив	Так	-	Позиції Петрі-об’єкта
transitions	Масив	Так	-	Переходи Петрі-об’єкта

Параметрі позиції наведено в таблиці 3.3.

Таблиця 3.3 — Параметри позиції

Назва параметра	Тип параметра	Обов’язковий	Значення за замовчуванням	Опис параметра
placeId	Ціле додатне число	Так	-	Ідентифікатор позиції
placeName	Строка	Так	-	Назва позиції
marking	Ціле невід’ємне число	Ні	0	Початкове маркування позиції
collectStatistics	Булева змінна	Ні	false	Позначення, чи потрібно підраховувати статистику для цієї позиції

Параметри переходу наведено в таблиці 3.4

Таблиця 3.4 — Параметри переходу

Назва параметра	Тип параметра	Обов’язковий	Значення за замовчуванням	Опис параметра
transitionId	Ціле додатне число	Так	-	Ідентифікатор переходу
transition Name	Строка	Так	-	Назва переходу
priority	Ціле невід’ємне число	Ні	0	Пріоритет запуску переходу
probability	Дробове число між 0 та 1	Ні	1	Ймовірність запуску переходу
delay Generator	JSON об’єкт з параметрами генератора затримки	Так	-	Параметри генератора затримки
collect Statistics	Булева змінна	Ні	false	Позначення, чи потрібно підраховувати статистику для цього переходу
arcsIn	Масив	Так	-	Масив вхідних дуг
arcsOut	Масив	Так	-	Масив вихідних дуг

Параметри генератора затримки наведено в таблиці 3.5.

Таблиця 3.5 — Параметри генератора затримки

Назва параметра	Тип параметра	Обов’язковий	Значення за замовчуванням	Опис параметра
type	Строка “CONSTANT”, “EXPONENTIAL”, “NORMAL” чи “UNIFORM”	Так	-	Тип генератора затримки
firstParam	Дробове число	Так	-	Перший параметр в формулі відповідного типу генератора затримки
secondParam	Дробове число	Ні	0	Другий параметр в формулі відповідного типу генератора затримки

Параметри вхідних дуг наведено в таблиці 3.6.

Таблиця 3.6 — Параметри вхідних дуг

Назва параметра	Тип параметра	Обов’язковий	Значення за замовчуванням	Опис параметра
placeId	Ціле додатне число	Так	-	Ідентифікатор позиції, з якої виходить дуга
multiplicity	Ціле додатне число	Ні	1	Кратність дуги
informational	Булева змінна	Ні	false	Покажчик інформаційна це дуга чи ні

Параметри вихідних дуг наведено в таблиці 3.7

Таблиця 3.7 — Параметри вихідних дуг

Назва параметра	Тип параметра	Обов’язковий	Значення за замовчуванням	Опис параметра
placeId	Ціле додатне число	Так	-	Ідентифікатор позиції, в яку входить дуга
multiplicity	Ціле додатне число	Ні	1	Кратність дуги

Параметри зовнішніх дуг наведено в таблиці 3.8

Таблиця 3.8 — Параметри зовнішніх дуг

Назва параметра	Тип параметра	Обов’язковий	Значення за замовчуванням	Опис параметра
fromPetriObjectId	Ціле додатне число	Так	-	Ідентифікатор Петрі-об’єкта, з якого виходить дуга
fromTransitionId	Ціле додатне число	Так	-	Ідентифікатор переходу, з якого виходить дуга
toPetriObjectId	Ціле додатне число	Так	-	Ідентифікатор Петрі-об’єкта, в який входить дуга
toPlaceId	Ціле додатне число	Так	-	Ідентифікатор позиції, в яку входить дуга
multiplicity	Ціле додатне число	Ні	1	Кратність дуги

3.3.2 Формат вихідних даних

Вихідні дані містять асоціативний масив, ключ у якому — ідентифікатор Петрі-об’єкта, а значення — статистика Петрі-об’єкта. Статистика Петрі-об’єкта містить два асоціативні масиви: transitionsStatistics і placesStatistics. Їх ключі — ідентифікатори переходів і позицій відповідно, а значення — статистика переходів і позицій відповідно. Для переходів збирається статистика кінцевого розміру буфера, максимального розміру буфера, мінімального

розміру буфера та середнього розміру буфера. Для позицій збирається статистика кінцевого маркування, максимального маркування, мінімального маркування та середнього маркування.

3.4 Висновки до розділу

Третій розділ концентрується на розробці програмного забезпечення. Розглянуті основні особливості кожної з чотирьох складових системи: розподілений алгоритм в вузлових серверах, принцип роботи сервісу повідомлень, робота з WebSocket та використання алгоритму round-robin в основному сервісі та робота з WebSocket в веб-інтерфейсі.

Розділ розглядає й приклад роботи з розробленим сервісом, описує можливі параметри, які сервіс приймає на вхід та структуру відповідей сервісу.

4 РОЗРОБКА БІЗНЕС-ПЛАНУ ПРОЄКТУ

4.1 Опис ідеї проєкту

В таблиці 4.1 розглянуто основну ідею проєкту та вигоди користувача від неї за напрямками застосування.

Таблиця 4.1 – Опис ідеї системи паралельної імітації дискретно-подійних систем

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Розробити сервіс, що буде паралельно імітувати Петрі-об'ємні моделі та розподіллено імітувати надвеликі моделі	Дослідження дискретно-подійних систем (виробництва, інфраструктура, мережі та інші)	Можливість запуску імітації без потреби купляти чи орендувати ресурси для обчислення
		Швидкий запуск імітації дискретно-подійних моделей
		Можливість імітувати надвеликі дискретно-подійні моделі

Розроблену ідею необхідно порівняти із конкурентами, що вже є на ринку імітації моделей, порівняти основні властивості та визначити слабкі та сильні сторони проєкту. Зроблене порівняння наведено в таблиці 4.2.

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик систему паралельної імітації дискретно-подійних систем

№	Техніко-економічні характеристики ідеї	Товари конкурентів			W	N	S
		Мій проєкт	AnyLogic	Arena			
1	Імітація моделей у вигляді систем масового обслуговування	Відсутня, але аналогічну модель можна створити, використовуючи Петрі-об’єктні моделі	Доступна. Спрощує розуміння моделі	Доступна. Спрощує розуміння моделі		+	
2	Імітація моделей у вигляді Петрі-об’єктної мережі	Доступна. Дозволяє гнучко створювати моделі	Відсутня	Відсутня			+
3	Анімація імітації	Відсутня	Доступна. Спрощує розуміння моделі	Доступна. Спрощує розуміння моделі	+		

Продовження таблиці 4.2

4	Ресурс, на якому запускається проєкт	Запускається на серверах, користувач не потребує додаткових програм	Встановлюється на Windows, MacOS чи Linux	Встановлюється на Windows			+
5	Інтеграція	Відсутня	Можна інтегруватися в процес імітації за допомогою мови Java	Можна інтегруватися в процес імітації за допомогою мови SIMAN	+		
6	Імітація надвеликих моделей	+	-	-			+

Сервіс паралельної імітації дискретно-подійних систем має свої недоліки у порівнянні з конкурентами, але його переваги такі вагомі, що робить сервіс однозначно конкурентоспроможним.

4.2 Технологічний аудит ідеї проєкту

Розглянемо технології, необхідні для створення сервісу паралельної імітації дискретно-подійних систем. Аналіз їх наявності та доступності наведено в таблиці 4.3.

Таблиця 4.3 – Технологічна здійсненність сервісу паралельної імітації дискретно-подійних систем

Ідея проєкту	Технології реалізації	Наявність технології	Доступність технології
Сервіс паралельної імітації дискретно-подійних систем з можливістю розподіленої імітації	Алгоритм паралельної імітації Петрі-об’єктних моделей	Технологія наявна	Технологія доступна
	Алгоритм розподіленої імітації Петрі-об’єктних моделей	Розроблена у рамках проєкту	Розроблена у рамках проєкту
	JDK8	Технологія наявна	Технологія доступна
	Spring Boot 2	Технологія наявна	Технологія доступна
	RabbitMQ	Технологія наявна	Технологія доступна
	JavaScript	Технологія наявна	Технологія доступна
Обрані технології реалізації проєкту: алгоритми паралельної та послідовної імітації Петрі-об’єктних моделей, JDK8, Spring Boot 2, RabbitMQ, JavaScript			

Технологічна реалізація проєкту можлива.

4.3 Аналіз ринкових можливостей запуску проєкту

В таблиці 4.4 розглянуто аналіз ринку проєктів імітації дискретно-подійних систем.

Таблиця 4.4 – Характеристика потенційного ринку сервісу паралельних обчислень дискретно-подійних систем

№	Показники стану ринку	Характеристика
1	Кількість головних гравців	Десятки
2	Динаміка ринку	Стагнує
3	Наявність обмежень для входу	Консервативність ринку, наявність невеликої кількості дуже відомих та широко застосовуваних рішень
4	Специфічні вимоги до стандартизації та сертифікації	Відсутні
5	Середня норма рентабельності в галузі	~25%

За попереднім оцінюванням проєкт є привабливим для входження, оскільки рентабельність вища за банківський відсоток для вкладення.

Розглянемо характеристики потенційних клієнтів, щоб краще орієнтуватися в ринку імітації та мати на що опиратися при визначенні стратегії розвитку сервісу (Таблиця 4.5).

Таблиця 4.5 – Потенційні клієнти сервісу

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Виконувати імітацію для оптимізації підприємства	Дослідники, що працюють на підприємство	Потребують можливості швидко приймати рішення на основі результатів імітації	Швидкість імітації

Продовження таблиці 4.5

2	Виконувати імітацію для оптимізації інфраструктури, транспорту та інші системи під відповідальністю держави	Дослідники, що обслуговують державу	Потребують рішення з чітко визначеною не великою вартістю	Вартість сервісу
3	Виконувати імітацію у наукових цілях	Науковці	Потребують імітацію високої точності з можливістю детальної розробки моделі	Гнучкість та точність імітації

З таблиці видно, що вимоги споживачів до товару співпадають з сильними сторонами сервісу паралельної імітації дискретно-подійних систем.

Проведемо аналіз середовища ринку імітації, розглянувши фактори загроз в таблиці 4.6 та фактори можливостей в таблиці 4.7.

Таблиця 4.6 – Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Незацікавленість клієнтів в новому інструменті	Користувачі можуть бути незацікавлені в новому для себе інструменті імітації, а нові користувачі вибиратимуть більш популярне рішення	Тісна співпраця з замовниками та клієнтами. Пояснення переваг переходу на наш сервіс, активна допомога користувачам на форумах та у соціальних мережах. Контент, що навчить користувачів та пояснить переваги

Продовження таблиці 4.6

2	Різке підвищення кількості користувачів	При різкому підвищенні кількості користувачів сервіси можуть сповільнюватися, що відверне велику кількість клієнтів	За рахунок мікросервісної архітектури можна збільшувати кількість найбільш навантажених модулів системи, орендуючи тимчасові ресурси під це розширення
3	Критичні помилки в роботі системи	Помилок при розробці не уникнути і при збільшенні кількості клієнтів може виявитися більше помилок у роботі системи	Розробити систему моніторингу сервісу та створити зручний спосіб клієнтам повідомляти про помилки в роботі. Постійно працювати над оновленням сервісу та впроваджувати регулярні зміни

Таблиця 4.7 – Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Зацікавлення інвесторів	Ідея може зацікавити інвесторів, через що до проєкту прийдуть гроші	Збільшення штату розробників, для швидшого виходу на ринок; розширення роботи з клієнтами
2	Співпраця з великим клієнтом	На ринку імітації є клієнти, що дуже активно користуються сервісами. Така компанія може зацікавитися співпрацею	Створення окремого штату з роботи з такими клієнтами, допомагати впроваджувати рішення в компанії, розробляти деякі модулі систему окремо під великих клієнтів

Щоб ще детальніше зрозуміти ринок, необхідно проаналізувати конкуренцію на ринку імітації. Аналіз конкуренції наведено у наступних двох таблицях: ступеневий аналіз в таблиці 4.8 та аналіз за Майклом Портером в таблиці 4.9.

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку імітації

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства
1. Чиста конкуренція	Присутність на ринку багатьох гравців, що конкурують	Дозволяє вийти на ринок, створивши унікальну пропозицію
2. Міжнаціональна конкуренція	Усі користувачі на ринку з різних країн	Підприємство повинно орієнтуватися на якнайширшу аудиторію
3. Внутрішньогалузева конкуренція	Конкуренція лише між сервісами імітації	Активніше розробляти сильні сторони сервісу у порівнянні з конкурентами
4. Товарно-родова конкуренція	Сервіси імітації можуть бути у різних формах (нативний додаток, веб-додаток та інші)	Працювати в напрямку веб-додатку, як найдоступнішому та найпопулярнішому
5. Цінова конкуренція	Послуги сервісів імітації чи самі додатки імітації платні	Зменшувати ціну для користувача за рахунок імітації як сервіс
6. Марочна конкуренція	Конкуренти пропонують схожий продукт (імітацію) за максимально низькими цінами	Зменшувати ціну для користувача за рахунок імітації як сервіс

Таблиця 4.9 – Аналіз конкуренції в галузі імітації за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти
	The AnyLogic Company, Rockwell Automation, SIMUL8 Corporation, ESI ITI GmbH та інші корпорації	На ринок не часто виходять нові конкуренти	Хмарні сервіси, між якими досить просто переходити	Успіх проєкту повністю залежить від великої кількості або від великих клієнтів
Висновки:	Конкуренти давно на ринку та їх продукти набули великих розмірів	Потенційних конкурентів немає сенсу розглядати	Постачальники майже не впливають на проєкт	Проєкт має бути повністю клієнто-орієнтованим

З огляду на аналіз конкуренції можна зробити висновок, що конкуренція на ринку можлива та проєкт може конкурувати. Для успішної конкуренції виділено основні переваги проєкту у таблиці 4.10.

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування
1	Гнучкість інструменту створення моделей для імітації	Задання Петрі-об’єктних моделей на вхід імітації гарантує високий рівень гнучкості
2	Ресурси, що виконують імітацію	Імітація запускається у хмарних сервісах, а не на комп’ютерах користувачів
3	Імітація надвеликих моделей	Алгоритм розподіленої імітації дискретно-подійних моделей дозволяє імітувати надвеликі моделі

Продовження таблиці 4.10

4	Ціна імітації	Клієнт платить за сервіс лише коли потребує його; нема необхідності купляти цілий програмний додаток

За виділеними факторами сервіс паралельної імітації дискретно-подійних систем випереджує конкурентів, що наведено в таблиці 4.11.

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін сервісу паралельної імітації дискретно-подійних моделей

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні						
			-3	-2	-1	0	+1	+2	+3
1	Гнучкість інструменту створення моделей для імітації	19		+					
2	Ресурси, що виконують імітацію	20		+					
3	Імітація надвеликих моделей	18	+						
4	Ціна імітації	19			+				

У якості фінального етапу аналізу можливостей сервісу на ринку імітації розроблено SWOT-аналіз у таблиці 4.12.

Таблиця 4.12 – SWOT-аналіз сервісу паралельної імітації дискретно-подійних моделей

Сильні сторони: Гнучкість інструменту створення моделей для імітації; Ресурси, що виконують імітацію; Імітація надвеликих моделей; Ціна імітації.	Слабкі сторони: Конкуренти - великі корпорації, що давно на цьому ринку; Відсутня інтеграція в процес імітації; Складне розуміння Петрі-об’єктних моделей

Продовження таблиці 4.12

Можливості: Горизонтальне масштабування системи; Співпраця з великими клієнтами; Розширення штабу працівників	Загрози: Незацікавленість інвесторів; Занадто різке зростання кількості користувачів; Активізація конкурентів з виходом нового рішення на ринок.

На основі SWOT-аналізу розроблено перелік заходів для виведення сервісу паралельної імітації дискретно-подійних систем на ринок у таблиці 4.13.

Таблиця 4.13 – Альтернативи ринкового впровадження проєкту

№	Альтернатива ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Розробка основного функціоналу на основі напрацювань	80%	6 місяців
2	Розробка функціоналу поступово, ітеративно	80%	10 місяців
3	Зміна типу використовуваних дискретно-подійних моделей	40%	6 місяців
4	Маркетингова компанія, що пояснить споживачам переваги	85%	12 місяців

4.4 Розроблення ринкової стратегії проєкту

Для розроблення ринкової стратегії в першу чергу було проаналізовано цільові групи сервісу. Опис та вибір цільових груп наведено в таблиці 4.14.

Таблиця 4.14 – Вибір цільових груп потенційних споживачів

№	Цільова група	Готовність сприйняти продукт	Орієнтовний попит	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Науковці та дослідники	Готові	Середній	Висока	Потребують дуже гнучкого та точного рішення, але з малою ймовірністю змінюють звичний інструмент. Середня складність входу
2	Малий бізнес	Не готові	Низький	Низька	Висока складність входу. Малий бізнес не потребує імітації своїх систем
3	Середній бізнес	Готові	Високий	Середня	Простий вхід. Бізнес тільки починає проводити аналіз своїх систем, відкритий до нових інструментів

Продовження таблиці 4.14

4	Великий бізнес	Готові	Високий	Висока	Середня складність входу. Великий бізнес потребує імітації, але ймовірно вже використовує інші інструменти
5	Державні установи	Готові	Середній	Середня	Висока складність входу. Потребує особливих процесів
Обрано цільові групи: №1, №3, №4					

Враховуючі обрані цільові групи, в таблиці 4.15 розроблена базова стратегія розвитку.

Таблиця 4.15 – Базова стратегія розвитку

№	Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Вихід на ринок з базовим функціоналом. Маркетингова компанія, направлена на середній та великий бізнес. Приваблива політика ціноутворення	Маркетингова компанія, що розкриває суттєві переваги сервісу для середнього та великого бізнесу	Гнучкість інструменту створення моделей для імітації. Імітація на ресурсах сервісу. Імітація надвеликих моделей. Ціна імітації.	Стратегія диферен-ціації

За тим для ринкової стратегії було визначено базову стратегії конкурентної поведінки. Результат наведено в таблиці 4.16.

Таблиця 4.16 – Визначення базової стратегії конкурентної поведінки

№	Чи є проєкт «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1	Ні	Компанія буде шукати нових споживачів в сегменті середнього бізнесу та забирати існуючих у конкурентів серед великого бізнесу	Основні характеристики компанії оригінальні, але компанія буде копіювати конкурентів для простішого виходу на ринок	Стратегія заняття конкурентної ніші

Базова стратегія розвитку та базова стратегія конкурентної поведінки дозволила визначити стратегію позиціонування (таблиця 4.17).

Таблиця 4.17 – Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції	Асоціації, які мають сформувати комплексну позицію
1	Імітація на ресурсах сервісу	Диференціація	Оскільки проєкт побудований як веб-сервіс, користувачу не потрібно імітувати на власних чи орендованих ресурсах	Імітація на ресурсах сервісу; Гнучкість задання моделі; Імітація надвеликих моделей
2	Ціна імітації	Лідерство по витратах	Зниження ціни імітації за рахунок імітації за потребою	
3	Гнучкість задання моделі	Диференціація	Проєкт гарантує високий рівень гнучкості за рахунок використання Петрі-об’єктних моделей	
4	Імітація надвеликих моделей	Спеціалізація	Розподілений алгоритм дозволяє проєкту імітувати надвеликі моделі	

В визначених стратегіях вкладена система рішень, на яку повинен орієнтуватися проєкт при розвитку, виділені основні позиції та напрямки, якими повинен рухатися сервіс.

4.5 Розроблення маркетингової програми проєкту

Перш ніж розробляти маркетингову програму, в таблиці 4.18 визначена маркетингова концепція сервісу.

Таблиця 4.18 – Визначення ключових переваг концепції

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
1	Імітація на ресурсах сервісу	Економія коштів на свої чи орендовані ресурси	Користувачу не потрібно встановлювати додаток на свої чи орендовані ресурси
2	Гнучкість задання моделі	Імітація неординарних, складних моделей	Використання Петрі-об’єктних моделей
3	Імітація надвеликих моделей	Можливість за адекватний час імітувати надвеликі моделі	Використання розподіленого алгоритму імітації

В таблиці 4.19 уточняється ідея сервісу, описано його складові та організацію надання, що формує трирівневу маркетингову модель товару.

Таблиця 4.19 – Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="2">Сутність та складові</td></tr>
<tr><td>I. Товар за задумом</td><td colspan="2">Сервіс імітації дискретно-подійних моделей, що пришвидшує імітацію за рахунок паралельної імітації та підтримує імітацію надвеликих моделей</td></tr>
<tr><td rowspan="5">II. Товар у реальному виконанні</td><td>Властивість/характеристика</td><td>Доступність</td></tr>
<tr><td>1. Імітація систем
2. Імітація надвеликих систем</td><td>1. З браузера чи через API
2. З браузера чи через API</td></tr>
<tr><td colspan="2">Якість: інтеграційні тести, системні тести</td></tr>
<tr><td colspan="2">Пакування: не потребує</td></tr>
<tr><td colspan="2">Марка: назва системи</td></tr>
</table>

Продовження таблиці 4.19

III. Товар із підкріпленням	До продажу: базові можливості для невеликих моделей
	Після продажу: підтримка, надвеликі моделі, пришвидшена імітація

Однією з ключовою позицією сервісу є зниження ціни імітації за рахунок надання її у вигляді сервісу. Визначену цінову межу наведено в таблиці 4.20.

Таблиця 4.20 - Визначення меж встановлення ціни

№	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	0$ - 600$/місяць	0$ - 600$/місяць	>10000$	0-100$/місяць

Маркетингова стратегія повинна включати в себе систему збуту. Система збуду сервісу досить проста і наведена в таблиці 4.21.

Таблиця 4.21 - Формування системи збуту

№	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Закуповують послуги онлайн або особливими актами	Надавати доступ до сервісу	Прямий	Збут через веб-сайт та індивідуальна робота з державними установами

Беручи до увагу специфіку цільової аудиторії та позиціонування сервісу розроблено концепцію маркетингової комунікації (таблиця 4.22).

Таблиця 4.22 - Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Цільовий клієнт шукає найдешевше оптимальне рішення проблеми імітації	Соцмережі, пошукові системи, форуми, власний штат	Імітація на ресурсах сервісу; Гнучкість задання моделі; Імітація надвеликих моделей	Донести до цільової аудиторії переваги сервісу, пояснити потребу в імітації	Рекламне звернення направлене безпосередньо на потенційних клієнтів та виконує своє завдання

4.6 Висновки до розділу

В п’ятому розділі розроблено бізнес-план проєкту. Проаналізовано ринок і виявлено, що хоч ринок і стагнує, рентабельність на ньому на достатньо високому рівні, щоб впроваджувати рішення.

Виділено три основні групи клієнтів, на які націлений проєкт: науковці та дослідники, середній бізнес, великий бізнес. Оскільки конкуренція чиста, то сервіс повинен надавати унікальні можливості. Серед переваг виділено: імітація на стороні та ресурсах сервісу, гнучкий спосіб задання моделі, не висока ціна за послуги та імітація надвеликих моделей, що робить проєкт конкурентноспроможним. З оглядом на це можна зробити висновок, що подальше розроблення та впровадження сервісу паралельної імітації дискретно-подійних систем доцільне.

ВИСНОВКИ

У даній магістерській дисертації розглянуто поняття дискретно-подійного моделювання та його основні формалізми: систему масового обслуговування і Петрі-об’єктні моделі. Аналіз дозволив визначити, що Петрі-об’єктні моделі краще підходить для розробки сервісу, що було завданням роботи, за рахунок більшої гнучкості та уніфікації. Розглянуто паралельний алгоритм імітації Петрі-об’єктних моделей, проведено ряд експериментів, що підтверджує оптимальність вибору саме цього алгоритму для сервісу, пояснене пришвидшення алгоритму в порівнянні з аналогічним послідовним алгоритмом.

За тим проаналізовані підходи до архітектури сервісів: монолітний підхід, сервісно-орієнтований та мікросервісний. Обґрунтовано доцільність використання мікросервісної архітектури для сервісу. Детально розглянуті шаблони архітектури, що використовуються для проєктування сервісу.

У роботі описано процес розробки компонентів системи та доопрацювання алгоритму паралельної імітації Петрі-об’єктних моделей для роботи з надвеликими моделями. Розглянуто приклад роботи з системою та описані параметри, що система приймає на вхід.

Також розроблено бізнес-план проєкту сервісу паралельної імітації дискретно-подійних моделей, проаналізовано ринок імітації, визначено цільові групи проєкту, досліджено конкуренцію на ринку та визначено переваги, що робить проєкт конкурентоспроможним.

Спроєктований і розроблений сервіс паралельної імітації дискретно-подійних систем має ряд переваг:

- імітація на ресурсах сервісу;
- гнучкість задання моделі для імітації;
- можливість імітувати надвеликі моделі.

За результатом даної магістерської дисертації удосконалено паралельний алгоритм імітації Петрі-об’єктних моделей для надвеликих моделей за рахунок впровадження його в розподіленій обчислювальній системі.

ПЕРЕЛІК ПОСИЛАНЬ

1) Barrett JS, Jayaraman B, Patel D, Skolnik J A SAS-based solution to evaluate study design efficiency of phase I pediatric oncology trials via discrete event simulation [Електронний ресурс] // Режим доступу: https://www.med.upenn.edu/kmas/DES.htm

2) Borshchev, A. (2013) The Big Book of Simulation Modeling. AnyLogic, America, 612.

3) Dennis Kafura Notes on Petri Nets // Computational Thinking [Електронний ресурс] // Режим доступу: http://people.cs.vt.edu/kafura/ComputationalThinking/Class-Notes/Petri-Net-Notes-Expanded.pdf

4) Рудольф Мачадо «Monolithic Architecture» / Рудольф Мачадо – К. : 1995.

5) Pramodh Katkoori Application Architecture: Monolithic VS SOA VS Microservices – 2019 - [Електронний ресурс] // Режим доступу: https://www.whishworks.com/blog/mulesoft/monolithic-soa-microservices/

6) Romana Gnatyk Microservices vs Monolith: which architecture is the best choice for your business – 2018 - [Електронний ресурс] // Режим доступу: https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/

7) What is SOA [Електронний ресурс] // Режим доступу: https://www.educba.com/what-is-soa/

8) Siraj ul Haq Introduction to Monolithic Architecture and MicroServices Architecture [Електронний ресурс] // Режим доступу: https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63

9) Saad Arshed Monolithic vs SOA vs Microservices — How to Choose Your Application Architecture [Електронний ресурс] // Режим доступу:

https://medium.com/@saad_66516/monolithic-vs-soa-vs-microservices-how-to-choose-your-application-architecture-1a33108d1469

10) Samarpit Tuli Microservices vs SOA: What's the Difference [Електронний ресурс] // Режим доступу: https://dzone.com/articles/microservices-vs-soa-whats-the-difference

11) Madhuka Udantha Microservice Architecture and Design Patterns for Microservices [Електронний ресурс] // Режим доступу: https://medium.com/@madhukaudantha/microservice-architecture-and-design-patterns-for-microservices-e0e5013fd58a

12) Decompose by business capability [Електронний ресурс] // Режим доступу: https://microservices.io/patterns/decomposition/decompose-by-business-capability.html

13) Rajesh Bhojwani Design Patterns for Microservices – 2018 - [Електронний ресурс] // Режим доступу: https://dzone.com/articles/design-patterns-for-microservices

14) Event Sourcing pattern – 2017 - [Електронний ресурс] // Режим доступу: https://docs.microsoft.com/en-us/azure/architecture/patterns/event-sourcing

15) Педоренко, А. В. Бібліотека паралельних обчислень Петрі-об’єктних моделей : дипломний проєкт бакалавра : 6.050103 Програмна інженерія / Педоренко Андрій Вікторович. – Київ, 2019. – 115 с.

16) Стеценко І.В. Паралельний алгоритм імітації Петрі-об’єктної моделі / Інна Вячеславівна Стеценко. // Кібернетика і системний аналіз. - 2017. - №53(4). - С. 130-140.

17) Stetsenko Inna V., Dorosh Vitaliy I., Dyfuchyn Anton Petri-object sumulation: software package and compexite // Intelligent Data Acquisition and Andvanced Computing Systems: Technology and Applications (IDAACS), 2015 IEEE 8th International Conference. - IEEE, 2015. - Vol.1. - С. 381-385.

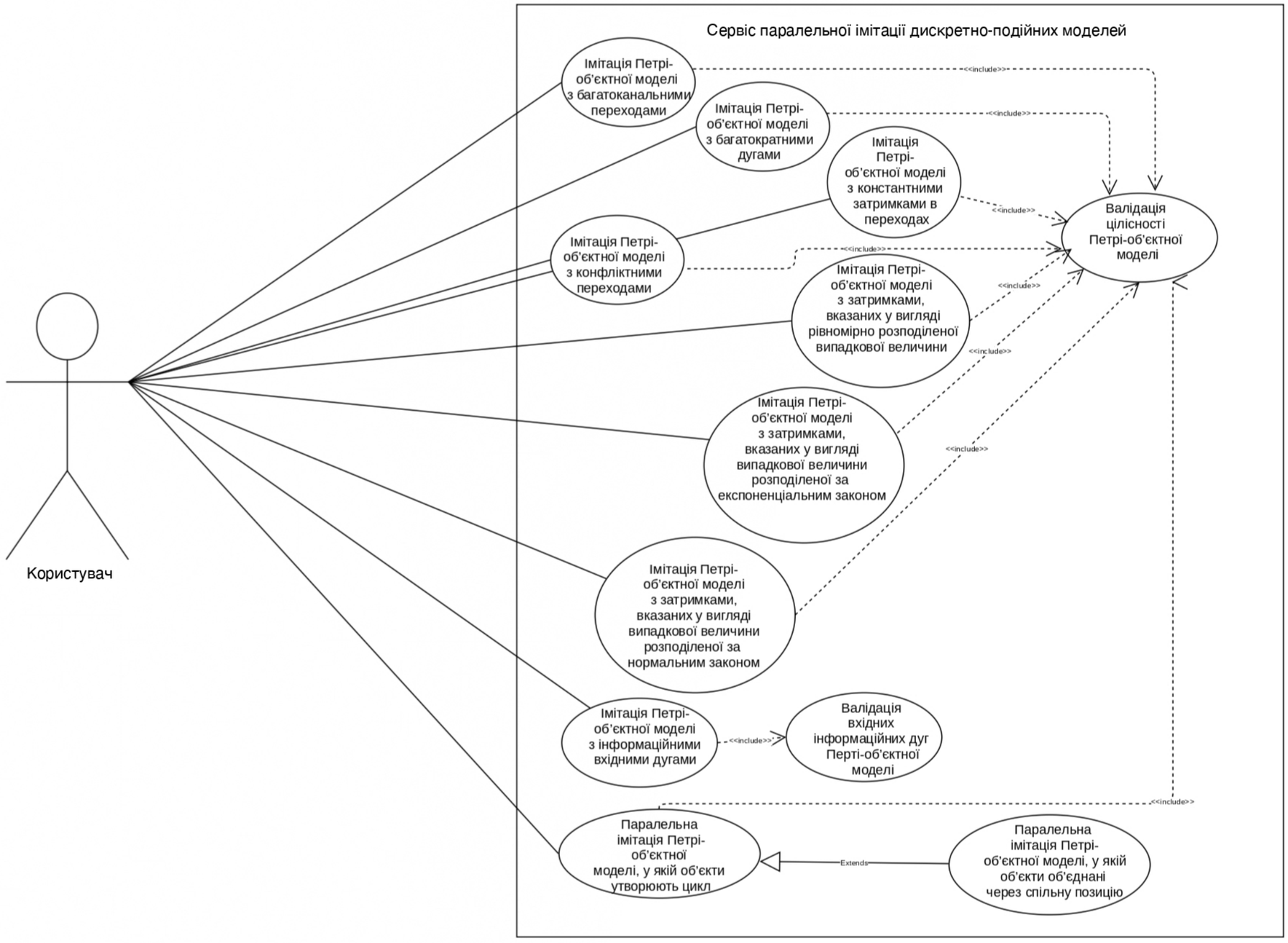
18) Lovisa Johansson What is RabbitMQ – 2019 - Електронний ресурс] // Режим доступу: https://www.cloudamqp.com/blog/2015-05-18-part1-rabbitmq-for-beginners-what-is-rabbitmq.html

19) HTML5 WebSocket: A Quantum Leap in Scalability for the Web [Електронний ресурс] // Режим доступу: http://www.websocket.org/quantum.html

20) Arpaci-Dusseau, Remzi H.; Arpaci-Dusseau, Andrea C., Operating Systems: Three Easy Pieces – 2014.

ДОДАТОК А

ГРАФІЧНІ МАТЕРІАЛИ



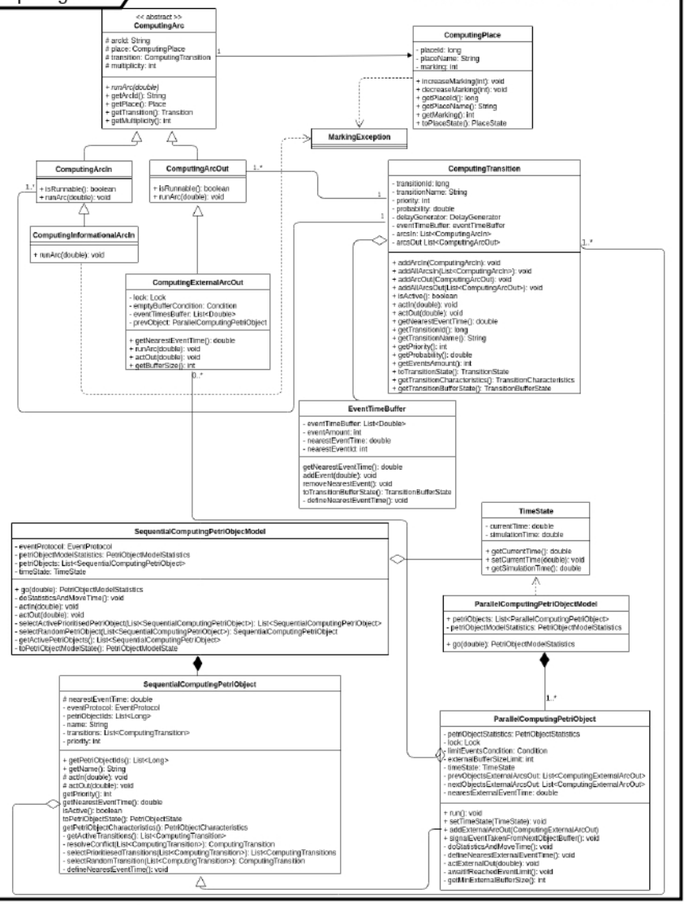
Демонстраційний плакат до магістерської дисертації

Схема структурна варіантів використань

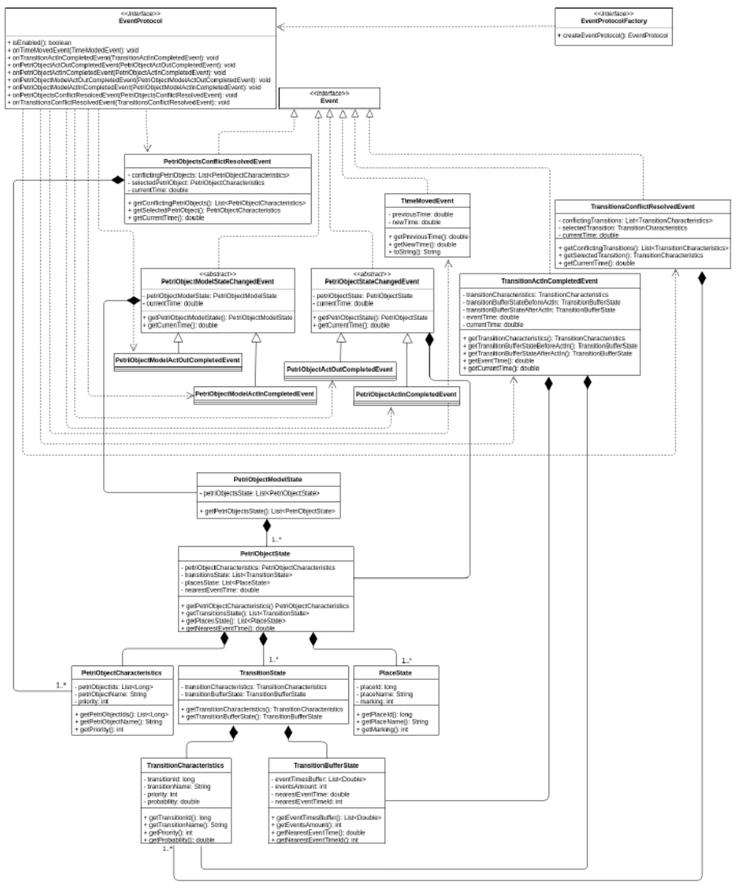
Виконав студент гр. ІП-92мп — Педоренко А.В.

Керівник — Стеценко І.В.

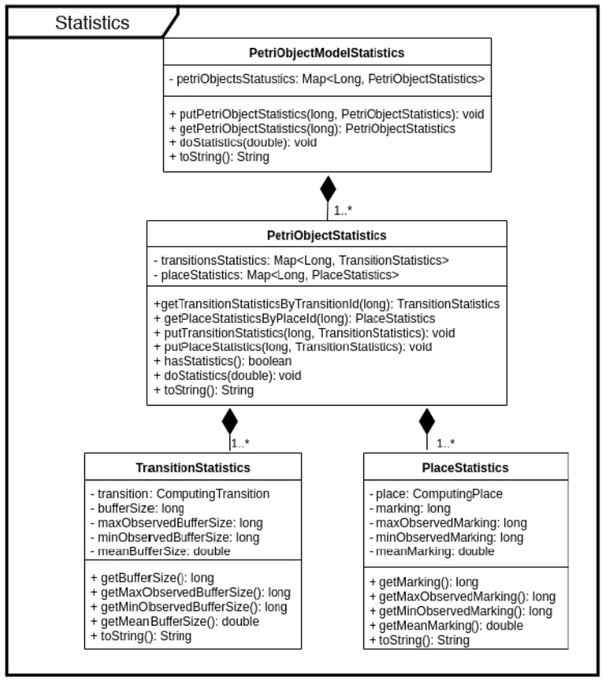
Computing model



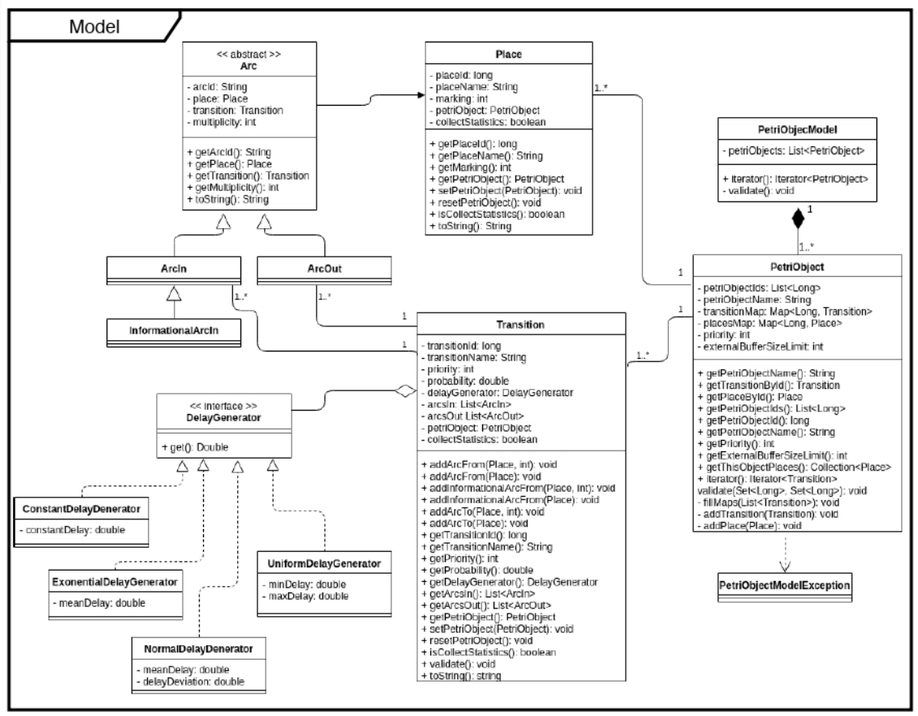
Event protocol



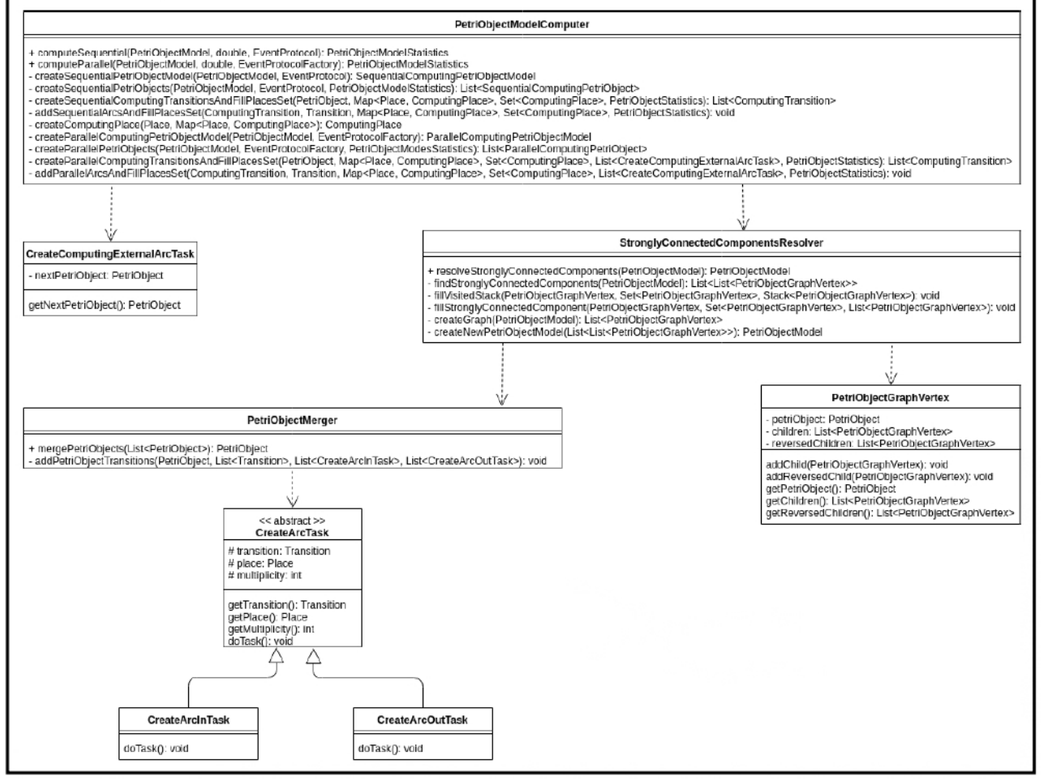
Statistics



Model



Model computer



Демонстраційний плакат до магістерської дисертації

Схема структурна класів вузлового сервісу

Виконав студент гр. ІП-92мп — Педоренко А.В.

Керівник — Стеценко І.В.

ДОДАТОК Б
ЛІСТИНГ ПРОГРАМИ

index html

```
<!DOCTYPE html>
<html>
<head>
    <title>Petri</title>
    <link href="/css/bootstrap.min.css" rel="stylesheet">
    <link href="/css/main.css" rel="stylesheet">
    <script src="/js/jquery.min.js"></script>
    <script src="/js/sockjs.min.js"></script>
    <script src="/js/stomp.min.js"></script>
    <script src="/js/app.js"></script>
</head>
<body>
<noscript><h2 style="color: #ff0000">Seems your browser doesn't support Javascript! Websocket relies on Javascript being
    enabled. Please enable
    Javascript and reload this page!</h2></noscript>
<div id="main-content" class="container">
    <div class="row">
        <div class="col-6">
            <form>
                <div class="d-flex flex-column">
                    <label for="net">Petri-object model JSON goes here</label>
                    <textarea id="net" rows="50" cols="100">

                    </textarea>
                    <button id="submit-net" class="btn btn-default" type="submit">Calculate</button>
                </div>
            </form>
        </div>
        <div class="col-6">
            <div class="d-flex flex-column">
                <label for="stats">Run statistics will appear here</label>
                <textarea id="stats" rows="50" cols="100"></textarea>
            </div>
        </div>
    </div>
</div>
</body>
</html>
```

app.js

```
var stompClient = null;
connect();
function connect() {
    var socket = new SockJS('/websocket');
    stompClient = Stomp.over(socket);
    stompClient.connect({}, function (frame) {
        console.log('Connected: ' + frame);
        stompClient.subscribe('/user/topic/statistics', function (statistics) {
            statsJson = JSON.parse(statistics.body);
            if (statsJson.message !== undefined) {
                showStatistics(statsJson.message);
            } else {
                showStatistics(JSON.stringify(JSON.parse(statistics.body), null, '\t'));
            }
        });
    });
}
function disconnect() {
    if (stompClient !== null) {
```

```
            stompClient.disconnect();
        }
        console.log("Disconnected");
}
function sendNet() {
        stompClient.send("/petri/compute/distributed", {}, $("#net").val());
}
function showStatistics(statistics) {
        $("#stats").val(statistics);
}
$(function () {
        $("form").on('submit', function (e) {
            e.preventDefault();
        });
        $( "#submit-net" ).click(function() { sendNet(); });
});
```

PetriApplication.java

```
package edu.pedorenko.petri;
import org.modelmapper.ModelMapper;
import org.modelmapper.convention.MatchingStrategies;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Import;
import javax.validation.Validator;
@SpringBootApplication
@Import(RabbitAutoConfiguration.class)
public class PetriApplication {
    public static void main(String[] args) {
        SpringApplication.run(PetriApplication.class, args);
    }
    @Bean
    public ModelMapper modelMapper() {
        ModelMapper modelMapper = new ModelMapper();
        modelMapper.getConfiguration().setMatchingStrategy(MatchingStrategies.STANDARD);
        return modelMapper;
    }
}
```

RabbitConfiguration.java

```
package edu.pedorenko.petri.config;
import org.springframework.amqp.core.AmqpAdmin;
import org.springframework.amqp.core.Queue;
import org.springframework.amqp.rabbit.connection.CachingConnectionFactory;
import org.springframework.amqp.rabbit.connection.ConnectionFactory;
import org.springframework.amqp.rabbit.core.RabbitAdmin;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.amqp.support.converter.Jackson2JsonMessageConverter;
import org.springframework.amqp.support.converter.MessageConverter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
public class RabbitConfiguration {
        @Bean
        public ConnectionFactory connectionFactory() {
            return new CachingConnectionFactory("localhost");
        }
        @Bean
        public AmqpAdmin amqpAdmin() {
```

```
            return new RabbitAdmin(connectionFactory());
        }
        @Bean
        public MessageConverter jsonMessageConverter() {
            return new Jackson2JsonMessageConverter();
        }
        @Bean
        public RabbitTemplate rabbitTemplate() {
            final RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory());
            rabbitTemplate.setMessageConverter(jsonMessageConverter());
            rabbitTemplate.setReplyTimeout(-1);
            return rabbitTemplate;
        }
        @Bean
        public Queue petri1Queue() {
            return new Queue("petri1");
        }
        @Bean
        public Queue petri2Queue() {
            return new Queue("petri2");
        }
}
```

WebSocketConfig.java

```
package edu.pedorenko.petri.config;
import com.fasterxml.jackson.databind.DeserializationFeature;
import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.converter.MappingJackson2MessageConverter;
import org.springframework.messaging.converter.MessageConverter;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;
import org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigurer;
import java.util.List;
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {
        public void configureMessageBroker(MessageBrokerRegistry config) {
            config.enableSimpleBroker("/topic");
            config.setApplicationDestinationPrefixes("/petri");
        }
        public void registerStompEndpoints(StompEndpointRegistry registry) {
            registry.addEndpoint("/websocket").withSockJS();
        }
        public boolean configureMessageConverters(List<MessageConverter> messageConverters) {
            messageConverters.clear();
            MappingJackson2MessageConverter messageConverter = new MappingJackson2MessageConverter();
            messageConverter.getObjectMapper()
                        .enable(DeserializationFeature.ACCEPT_EMPTY_STRING_AS_NULL_OBJECT);
            messageConverters.add(messageConverter);
            return false;
        }
}
```

PetriWebSocketController.java

```
package edu.pedorenko.petri.controller;
import edu.pedorenko.petri.dto.PetriObjectModelDTO;
import edu.pedorenko.petri.dto.PreprocessedPetriObjectModelDTO;
import edu.pedorenko.petri.dto.statistics.PetriObjectModelStatisticsDTO;
import edu.pedorenko.petri.exception.ParallelExecutionException;
import edu.pedorenko.petri.exception.PetriObjectModelCreatingException;
```

```
import edu.pedorenko.petri.model.PetriObjectModel;
import edu.pedorenko.petri.model.PetriObjectModelException;
import edu.pedorenko.petri.service.PetriObjectModelCreatingService;
import edu.pedorenko.petri.service.PetriObjectsDistributedComputingService;
import edu.pedorenko.petri.util.cycles_resolver.StronglyConnectedComponentsResolver;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.converter.HttpMessageNotReadableException;
import org.springframework.messaging.converter.MessageConversionException;
import org.springframework.messaging.handler.annotation.MessageExceptionHandler;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.support.MethodArgumentNotValidException;
import org.springframework.messaging.simp.annotation.SendToUser;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import javax.validation.Valid;
import java.util.Date;
@Controller
public class PetriWebSocketController {
    private final PetriObjectModelCreatingService petriObjectModelCreatingService;
    private final PetriObjectsDistributedComputingService petriObjectsDistributedComputingService;
    @Autowired
    public PetriWebSocketController(
            PetriObjectModelCreatingService petriObjectModelCreatingService,
            PetriObjectsDistributedComputingService petriObjectsDistributedComputingService) {
        this.petriObjectModelCreatingService = petriObjectModelCreatingService;
        this.petriObjectsDistributedComputingService = petriObjectsDistributedComputingService;
    }
    @MessageMapping("/compute/distributed")
    @SendToUser("/topic/statistics")
    public PetriObjectModelStatisticsDTO computePetriObjectModelDistributed(@Valid @RequestBody PetriObjectModelDTO petriObjectModelDTO) {
        PetriObjectModel petriObjectModel = petriObjectModelCreatingService.createPetriObjectModel(petriObjectModelDTO);
        PetriObjectModel preprocessedPetriObjectModel;
        try {
            preprocessedPetriObjectModel = StronglyConnectedComponentsResolver.resolveStronglyConnectedComponents(petriObjectModel);
        } catch (PetriObjectModelException ex) {
            throw new PetriObjectModelCreatingException(ex.getMessage());
        }
        PreprocessedPetriObjectModelDTO preprocessedPetriObjectModelDTO =
                petriObjectModelCreatingService.createPreprocessedPetriObjectModelDTO(
                        preprocessedPetriObjectModel,
                        petriObjectModelDTO.getTimeModelling());
        return petriObjectsDistributedComputingService.computeDistributed(petriObjectModelDTO, preprocessedPetriObjectModelDTO);
    }
    @MessageExceptionHandler(
            {
                    PetriObjectModelCreatingException.class,
                    MethodArgumentNotValidException.class,
                    HttpMessageNotReadableException.class,
                    IllegalArgumentException.class,
                    MessageConversionException.class
            })
    @SendToUser("/topic/statistics")
    public JsonResponse handleCreatingExceptions(Exception ex) {
        return new PetriWebSocketController.JsonResponse(
                        new Date(),
                        HttpStatus.BAD_REQUEST.value(),
                        HttpStatus.BAD_REQUEST.getReasonPhrase(),
                        ex.getMessage());
    }
    @MessageExceptionHandler({ParallelExecutionException.class})
    @SendToUser("/topic/statistics")
    public JsonResponse handleParallelExecutionException(ParallelExecutionException ex) {
        return new PetriWebSocketController.JsonResponse(
                        new Date(),
```

```
                HttpStatus.INTERNAL_SERVER_ERROR.value(),
                HttpStatus.INTERNAL_SERVER_ERROR.getReasonPhrase(),
                ex.getMessage());
    }
    private class JsonResponse {
        Date timestamp;
        int status;
        String error;
        String message;
        public JsonResponse() {
        }
        public JsonResponse(Date timestamp, int status, String error, String message) {
            this.timestamp = timestamp;
            this.status = status;
            this.error = error;
            this.message = message;
        }
        public Date getTimestamp() {
            return timestamp;
        }
        public void setTimestamp(Date timestamp) {
            this.timestamp = timestamp;
        }
        public int getStatus() {
            return status;
        }
        public void setStatus(int status) {
            this.status = status;
        }
        public String getError() {
            return error;
        }
        public void setError(String error) {
            this.error = error;
        }
        public String getMessage() {
            return message;
        }
        public void setMessage(String message) {
            this.message = message;
        }
    }
}
```

ParallelExecutionException.java

```
package edu.pedorenko.petri.exception;
public class ParallelExecutionException extends RuntimeException {
    public ParallelExecutionException(String message) {
        super(message);
    }
}
```

PetriObjectModelCreatingException.java

```
package edu.pedorenko.petri.exception;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;
@ResponseStatus(value = HttpStatus.BAD_REQUEST)
public class PetriObjectModelCreatingException extends RuntimeException {
    public PetriObjectModelCreatingException(String message) {
        super(message);
    }
```

```
}
```

PetriObjectModelExceptionHandler.java

```
package edu.pedorenko.petri.exception;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.http.converter.HttpMessageNotReadableException;
import org.springframework.messaging.handler.annotation.MessageExceptionHandler;
import org.springframework.messaging.simp.annotation.SendToUser;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import java.util.Date;
@ControllerAdvice
public class PetriObjectModelExceptionHandler {
    @MessageExceptionHandler(
            {
                    PetriObjectModelCreatingException.class,
                    MethodArgumentNotValidException.class,
                    HttpMessageNotReadableException.class,
                    IllegalArgumentException.class
            })
    @SendToUser("/topic/statistics")
    public ResponseEntity<JsonResponse> handleCreatingExceptions(Exception ex) {
        return new ResponseEntity<>(
                new JsonResponse(
                        new Date(),
                        HttpStatus.BAD_REQUEST.value(),
                        HttpStatus.BAD_REQUEST.getReasonPhrase(),
                        ex.getMessage()),
                HttpStatus.BAD_REQUEST);
    }
    @MessageExceptionHandler({ParallelExecutionException.class})
    @SendToUser("/topic/statistics")
    public ResponseEntity<JsonResponse> handleParallelExecutionException(ParallelExecutionException ex) {
        return new ResponseEntity<>(
                new JsonResponse(
                        new Date(),
                        HttpStatus.INTERNAL_SERVER_ERROR.value(),
                        HttpStatus.INTERNAL_SERVER_ERROR.getReasonPhrase(),
                        ex.getMessage()),
                HttpStatus.INTERNAL_SERVER_ERROR);
    }
    private class JsonResponse {
        Date timestamp;
        int status;
        String error;
        String message;
        public JsonResponse() {
        }
        public JsonResponse(Date timestamp, int status, String error, String message) {
            this.timestamp = timestamp;
            this.status = status;
            this.error = error;
            this.message = message;
        }
        public Date getTimestamp() {
            return timestamp;
        }
        public void setTimestamp(Date timestamp) {
            this.timestamp = timestamp;
        }
        public int getStatus() {
```

```
            return status;
        }
        public void setStatus(int status) {
            this.status = status;
        }
        public String getError() {
            return error;
        }
        public void setError(String error) {
            this.error = error;
        }
        public String getMessage() {
            return message;
        }
        public void setMessage(String message) {
            this.message = message;
        }
    }
}
```

PetriObjectModelCreatingService.java

```
package edu.pedorenko.petri.service;
import edu.pedorenko.petri.dto.ArcDTO;
import edu.pedorenko.petri.dto.ArcInDTO;
import edu.pedorenko.petri.dto.DelayGeneratorDTO;
import edu.pedorenko.petri.dto.DelayGeneratorTypeDTO;
import edu.pedorenko.petri.dto.ExternalArcDTO;
import edu.pedorenko.petri.dto.PetriObjectDTO;
import edu.pedorenko.petri.dto.PetriObjectModelDTO;
import edu.pedorenko.petri.dto.PlaceDTO;
import edu.pedorenko.petri.dto.PreprocessedPetriObjectDTO;
import edu.pedorenko.petri.dto.PreprocessedPetriObjectModelDTO;
import edu.pedorenko.petri.dto.TransitionDTO;
import edu.pedorenko.petri.exception.PetriObjectModelCreatingException;
import edu.pedorenko.petri.model.PetriObject;
import edu.pedorenko.petri.model.PetriObjectModel;
import edu.pedorenko.petri.model.PetriObjectModelException;
import edu.pedorenko.petri.model.arc.ArcIn;
import edu.pedorenko.petri.model.arc.ArcOut;
import edu.pedorenko.petri.model.arc.InformationalArcIn;
import edu.pedorenko.petri.model.place.Place;
import edu.pedorenko.petri.model.transition.Transition;
import edu.pedorenko.petri.model.transition.delay_generator.ConstantDelayGenerator;
import edu.pedorenko.petri.model.transition.delay_generator.DelayGenerator;
import edu.pedorenko.petri.model.transition.delay_generator.ExponentialDelayGenerator;
import edu.pedorenko.petri.model.transition.delay_generator.NormalDelayGenerator;
import edu.pedorenko.petri.model.transition.delay_generator.UniformDelayGenerator;
import org.springframework.stereotype.Service;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
@Service
public class PetriObjectModelCreatingService {
    public PetriObjectModel createPetriObjectModel(PetriObjectModelDTO petriObjectModelDTO) {
        Map<Long, PetriObject> petriObjects = createPetriObjects(petriObjectModelDTO.getPetriObjects());
        for (ExternalArcDTO externalArcDTO : petriObjectModelDTO.getExternalArcs()) {
            PetriObject fromPetriObject = petriObjects.get(externalArcDTO.getFromPetriObjectId());
            PetriObject toPetriObject = petriObjects.get(externalArcDTO.getToPetriObjectId());
            if (fromPetriObject == null) {
                throw new PetriObjectModelCreatingException(
                        "No Petri object with id \"" + externalArcDTO.getFromPetriObjectId() + "\". Check external arcs");
            }
```

```
                if (toPetriObject == null) {
                        throw new PetriObjectModelCreatingException(
                                        "No Petri object with id \"" + externalArcDTO.getToPetriObjectId() + "\". Check external arcs");
                }
                Transition fromTransition = fromPetriObject.getTransitionById(externalArcDTO.getFromTransitionId());
                if (fromTransition == null) {
                        throw new PetriObjectModelCreatingException(
                                        "No transition with id \"" + externalArcDTO.getFromTransitionId() + "\"" +
                                                        " in Petri object with id \"" + externalArcDTO.getFromPetriObjectId() + "\"." +
                                                        " Check external arcs");
                }
                Place toPlace = toPetriObject.getPlaceById(externalArcDTO.getToPlaceId());
                if (toPlace == null) {
                        throw new PetriObjectModelCreatingException(
                                        "No place with id \"" + externalArcDTO.getToPlaceId() + "\"" +
                                                        " in Petri object with id \"" + externalArcDTO.getToPetriObjectId() + "\"." +
                                                        " Check external arcs");
                }
                fromTransition.addArcTo(toPlace, externalArcDTO.getMultiplicity());
        }
        try {
                return new PetriObjectModel(new ArrayList<>(petriObjects.values()));
        } catch (PetriObjectModelException ex) {
                throw new PetriObjectModelCreatingException(ex.getMessage());
        }
}
private Map<Long, PetriObject> createPetriObjects(List<PetriObjectDTO> petriObjectDTOs) {
        Map<Long, PetriObject> petriObjects = new HashMap<>();
        for (PetriObjectDTO petriObjectDTO : petriObjectDTOs) {
                PetriObject petriObject = createPetriObject(petriObjectDTO);
                petriObjects.put(petriObject.getPetriObjectId(), petriObject);
        }
        return petriObjects;
}
private PetriObject createPetriObject(PetriObjectDTO petriObjectDTO) {
        List<Transition> transitions = createTransitions(petriObjectDTO.getTransitions(), petriObjectDTO.getPlaces());
        try {
                return new PetriObject(petriObjectDTO.getPetriObjectId(), petriObjectDTO.getPetriObjectName(), transitions);
        } catch (PetriObjectModelException ex) {
                throw new PetriObjectModelCreatingException(ex.getMessage());
        }
}
private List<Transition> createTransitions(List<TransitionDTO> transitionDTOs, List<PlaceDTO> placeDTOs) {
        List<Transition> transitions = new ArrayList<>();
        Map<Long, Place> places = createPlaces(placeDTOs);
        for (TransitionDTO transitionDTO : transitionDTOs) {
                Transition transition = createTransition(transitionDTO, places);
                transitions.add(transition);
        }
        return transitions;
}
private Map<Long, Place> createPlaces(List<PlaceDTO> placeDTOs) {
        Map<Long, Place> places = new HashMap<>();
        for (PlaceDTO placeDTO : placeDTOs) {
                Place place = createPlace(placeDTO);
                if (places.containsKey(place.getPlaceId())) {
                        throw new PetriObjectModelCreatingException("Duplicate place id \"" + place.getPlaceId() + "\"");
                }
                places.put(place.getPlaceId(), place);
        }
        return places;
}
private Place createPlace(PlaceDTO placeDTO) {
        return new Place(
                        placeDTO.getPlaceId(),
```

```
                placeDTO.getPlaceName(),
                placeDTO.getMarking(),
                placeDTO.isCollectStatistics());
    }
    private Transition createTransition(TransitionDTO transitionDTO, Map<Long, Place> places) {
        DelayGenerator delayGenerator = createDelayGenerator(transitionDTO.getDelayGenerator());
        Transition transition = new Transition(
                transitionDTO.getTransitionId(),
                transitionDTO.getTransitionName(),
                transitionDTO.getPriority(),
                transitionDTO.getProbability(),
                delayGenerator,
                transitionDTO.isCollectStatistics());
        for (ArcInDTO arcDTO : transitionDTO.getArcsIn()) {
            Place fromPlace = places.get(arcDTO.getPlaceId());
            if (fromPlace == null) {
                throw new PetriObjectModelCreatingException(
                        "No place with id \"" + arcDTO.getPlaceId() + "\". " +
                                "Check arcs for transition with id \"" + transitionDTO.getTransitionId() + "\"");
            }
            if (arcDTO.isInformational()) {
                transition.addInformationalArcFrom(fromPlace, arcDTO.getMultiplicity());
            } else {
                transition.addArcFrom(fromPlace, arcDTO.getMultiplicity());
            }
        }
        for (ArcDTO arcDTO : transitionDTO.getArcsOut()) {
            Place toPlace = places.get(arcDTO.getPlaceId());
            if (toPlace == null) {
                throw new PetriObjectModelCreatingException(
                        "No place with id \"" + arcDTO.getPlaceId() + "\". " +
                                "Check arcs for transition with id \"" + transitionDTO.getTransitionId() + "\"");
            }
            transition.addArcTo(toPlace, arcDTO.getMultiplicity());
        }
        return transition;
    }
    private DelayGenerator createDelayGenerator(DelayGeneratorDTO delayGeneratorDTO) {
        DelayGeneratorTypeDTO delayGeneratorType = delayGeneratorDTO.getType();
        switch (delayGeneratorType) {
            case CONSTANT:
                return new ConstantDelayGenerator(delayGeneratorDTO.getFirstParam());
            case UNIFORM:
                return new UniformDelayGenerator(delayGeneratorDTO.getFirstParam(), delayGeneratorDTO.getSecondParam());
            case NORMAL:
                return new NormalDelayGenerator(delayGeneratorDTO.getFirstParam(), delayGeneratorDTO.getSecondParam());
            case EXPONENTIAL:
                return new ExponentialDelayGenerator(delayGeneratorDTO.getFirstParam());
            default:
                throw new PetriObjectModelCreatingException("Unrecognized delay generator type: " + delayGeneratorType.name());
        }
    }
    public PreprocessedPetriObjectModelDTO createPreprocessedPetriObjectModelDTO(
            PetriObjectModel preprocessedPetriObjectModel,
            double timeModelling) {
        Map<Long, List<ExternalArcDTO>> externalArcsDTOsMap = createExternalArcsDTOsMap(preprocessedPetriObjectModel);
        List<PreprocessedPetriObjectDTO> preprocessedPetriObjectDTOs =
                createPreprocessedPetriObjectDTOs(preprocessedPetriObjectModel, externalArcsDTOsMap, timeModelling);
        return new PreprocessedPetriObjectModelDTO(preprocessedPetriObjectDTOs);
    }
    private List<PreprocessedPetriObjectDTO> createPreprocessedPetriObjectDTOs(
            PetriObjectModel petriObjectModel,
            Map<Long, List<ExternalArcDTO>> externalArcsDTOsMap,
            double timeModelling) {
        List<PreprocessedPetriObjectDTO> preprocessedPetriObjectDTOs = new ArrayList<>();
```

```
        for (PetriObject petriObject : petriObjectModel) {
            List<ExternalArcDTO> externalArcDTOs = externalArcsDTOsMap.get(petriObject.getPetriObjectId());
            if (externalArcDTOs == null) {
                externalArcDTOs = new ArrayList<>();
            }
            PreprocessedPetriObjectDTO preprocessedPetriObjectDTO = createPreprocessedPetriObjectDTO(
                    petriObject,
                    externalArcDTOs,
                    timeModelling);
            preprocessedPetriObjectDTOs.add(preprocessedPetriObjectDTO);
        }
        return preprocessedPetriObjectDTOs;
    }
    private PreprocessedPetriObjectDTO createPreprocessedPetriObjectDTO(
            PetriObject petriObject,
            List<ExternalArcDTO> externalArcDTOs,
            double timeModelling) {
        List<Long> petriObjectIds = petriObject.getPetriObjectIds();
        String petriObjectName = petriObject.getPetriObjectName();
        List<PlaceDTO> placeDTOs = createPlaceDTOs(petriObject);
        List<TransitionDTO> transitionDTOs = createInnerTransitionDTOs(petriObject);
        return new PreprocessedPetriObjectDTO(
                petriObjectIds,
                petriObjectName,
                placeDTOs,
                transitionDTOs,
                externalArcDTOs,
                timeModelling);
    }
    private List<PlaceDTO> createPlaceDTOs(PetriObject petriObject) {
        List<PlaceDTO> placeDTOs = new ArrayList<>();
        for (Place place : petriObject.getThisObjectPlaces()) {
            PlaceDTO placeDTO = createPlaceDTO(place);
            placeDTOs.add(placeDTO);
        }
        return placeDTOs;
    }
    private PlaceDTO createPlaceDTO(Place place) {
        return new PlaceDTO(
                place.getPlaceId(),
                place.getPlaceName(),
                place.getMarking(),
                place.isCollectStatistics());
    }
    private List<TransitionDTO> createInnerTransitionDTOs(PetriObject petriObject) {
        List<TransitionDTO> innerTransitionDTOs = new ArrayList<>();
        for (Transition transition : petriObject) {
            TransitionDTO transitionDTO = createInnerTransitionDTO(transition);
            innerTransitionDTOs.add(transitionDTO);
        }
        return innerTransitionDTOs;
    }
    private TransitionDTO createInnerTransitionDTO(Transition transition) {
        long transitionId = transition.getTransitionId();
        String transitionName = transition.getTransitionName();
        int priority = transition.getPriority();
        double probability = transition.getProbability();
        DelayGeneratorDTO delayGeneratorDTO = createDelayGeneratorDTO(transition.getDelayGenerator());
        boolean collectStatistics = transition.isCollectStatistics();
        List<ArcInDTO> arcsInDTOs = createArcsIn(transition.getArcsIn());
        List<ArcDTO> innerArcsOutDTOs = createInnerArcsOut(transition.getArcsOut());
        return new TransitionDTO(
                transitionId,
                transitionName,
                priority,
```

```
                probability,
                delayGeneratorDTO,
                collectStatistics,
                arcsInDTOs,
                innerArcsOutDTOs);
    }
    private DelayGeneratorDTO createDelayGeneratorDTO(DelayGenerator delayGenerator) {
        if (delayGenerator instanceof ConstantDelayGenerator) {
            ConstantDelayGenerator constantDelayGenerator = (ConstantDelayGenerator) delayGenerator;
            return new DelayGeneratorDTO(
                    DelayGeneratorTypeDTO.CONSTANT,
                    constantDelayGenerator.getConstantDelay());
        }
        if (delayGenerator instanceof UniformDelayGenerator) {
            UniformDelayGenerator uniformDelayGenerator = (UniformDelayGenerator) delayGenerator;
            return new DelayGeneratorDTO(
                    DelayGeneratorTypeDTO.UNIFORM,
                    uniformDelayGenerator.getMinDelay(),
                    uniformDelayGenerator.getMaxDelay());
        }
        if (delayGenerator instanceof NormalDelayGenerator) {
            NormalDelayGenerator normalDelayGenerator = (NormalDelayGenerator) delayGenerator;
            return new DelayGeneratorDTO(
                    DelayGeneratorTypeDTO.NORMAL,
                    normalDelayGenerator.getMeanDelay(),
                    normalDelayGenerator.getDelayDeviation());
        }
        if (delayGenerator instanceof ExponentialDelayGenerator) {
            ExponentialDelayGenerator exponentialDelayGenerator = (ExponentialDelayGenerator) delayGenerator;
            return new DelayGeneratorDTO(
                    DelayGeneratorTypeDTO.EXPONENTIAL,
                    exponentialDelayGenerator.getMeanDelay());
        }
        throw new PetriObjectModelCreatingException("Unrecognized delay generator class: " + delayGenerator.getClass().getName());
    }
    private List<ArcInDTO> createArcsIn(List<ArcIn> arcsIn) {
        List<ArcInDTO> arcsInDTOs = new ArrayList<>();
        for (ArcIn arcIn : arcsIn) {
            ArcInDTO arcInDTO = createArcInDTO(arcIn);
            arcsInDTOs.add(arcInDTO);
        }
        return arcsInDTOs;
    }
    private ArcInDTO createArcInDTO(ArcIn arcIn) {
        long placeId = arcIn.getPlace().getPlaceId();
        int multiplicity = arcIn.getMultiplicity();
        boolean informational = arcIn instanceof InformationalArcIn;
        return new ArcInDTO(placeId, multiplicity, informational);
    }
    private List<ArcDTO> createInnerArcsOut(List<ArcOut> arcsOut) {
        List<ArcDTO> innerArcsOut = new ArrayList<>();
        for (ArcOut arcOut : arcsOut) {
            if (arcOut.getPlace().getPetriObject() == arcOut.getTransition().getPetriObject()) {
                ArcDTO arcDTO = createArcDTO(arcOut);
                innerArcsOut.add(arcDTO);
            }
        }
        return innerArcsOut;
    }
    private ArcDTO createArcDTO(ArcOut arcOut) {
        return new ArcDTO(
                arcOut.getPlace().getPlaceId(),
                arcOut.getMultiplicity());
    }
    private Map<Long, List<ExternalArcDTO>> createExternalArcsDTOsMap(PetriObjectModel petriObjectModel) {
```

```
        Map<Long, List<ExternalArcDTO>> externalArcDTOsMap = new HashMap<>();
        for (PetriObject petriObject : petriObjectModel) {
            for (Transition transition : petriObject) {
                for (ArcOut arcOut : transition.getArcsOut()){
                    PetriObject fromPetriObject = arcOut.getTransition().getPetriObject();
                    PetriObject toPetriObject = arcOut.getPlace().getPetriObject();
                    if (fromPetriObject != toPetriObject) {
                        ExternalArcDTO externalArcDTO = createExternalArcsDTO(arcOut);
                        addExternalArcDTOToMap(externalArcDTOsMap, externalArcDTO, fromPetriObject);
                        addExternalArcDTOToMap(externalArcDTOsMap, externalArcDTO, toPetriObject);
                    }
                }
            }
        }
        return externalArcDTOsMap;
    }
    private ExternalArcDTO createExternalArcsDTO(ArcOut arcOut) {
        Transition fromTransition = arcOut.getTransition();
        Place toPlace = arcOut.getPlace();
        return new ExternalArcDTO(
                fromTransition.getPetriObject().getPetriObjectId(),
                fromTransition.getTransitionId(),
                toPlace.getPetriObject().getPetriObjectId(),
                toPlace.getPlaceId(),
                arcOut.getMultiplicity());
    }
    private void addExternalArcDTOToMap(
            Map<Long, List<ExternalArcDTO>> externalArcDTOsMap,
            ExternalArcDTO externalArcDTO,
            PetriObject petriObject) {
        for (long petriObjectId : petriObject.getPetriObjectIds()) {
            List<ExternalArcDTO> externalArcDTOs = externalArcDTOsMap.computeIfAbsent(petriObjectId, list -> new ArrayList<>());
            externalArcDTOs.add(externalArcDTO);
        }
    }
}
```

PetriObjectsDistributedComputingService.java

```
package edu.pedorenko.petri.service;
import edu.pedorenko.petri.dto.PetriObjectModelDTO;
import edu.pedorenko.petri.dto.PlaceDTO;
import edu.pedorenko.petri.dto.PreprocessedPetriObjectDTO;
import edu.pedorenko.petri.dto.PreprocessedPetriObjectModelDTO;
import edu.pedorenko.petri.dto.TransitionDTO;
import edu.pedorenko.petri.dto.statistics.PetriObjectModelStatisticsDTO;
import edu.pedorenko.petri.dto.statistics.PetriObjectStatisticsDTO;
import edu.pedorenko.petri.dto.statistics.PlaceStatisticsDTO;
import edu.pedorenko.petri.dto.statistics.TransitionStatisticsDTO;
import edu.pedorenko.petri.exception.ParallelExecutionException;
import org.springframework.amqp.core.AmqpTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.stream.Collectors;
@Service
public class PetriObjectsDistributedComputingService {
```

```
        private final AmqpTemplate amqpTemplate;
        private final ExecutorService executorService = Executors.newWorkStealingPool();
        @Value("${petri.queue.names}")
        private String[] queuesNames;
        private AtomicInteger queueNamesCursor = new AtomicInteger(0);
        @Autowired
        public PetriObjectsDistributedComputingService(AmqpTemplate amqpTemplate) {
            this.amqpTemplate = amqpTemplate;
        }
        public PetriObjectModelStatisticsDTO computeDistributed(PetriObjectModelDTO petriObjectModelDTO, PreprocessedPetriObjectModelDTO preprocessedPetriObjectModelDTO) {
            Map<List<Long>, Future> futures = new HashMap<>();
            for (PreprocessedPetriObjectDTO petriObjectDTO : preprocessedPetriObjectModelDTO.getPetriObjects()) {
                final String queueName = getQueueName();
                futures.put(
                        petriObjectDTO.getPetriObjectIds(),
                        executorService.submit(() -> amqpTemplate.convertSendAndReceive(queueName, petriObjectDTO))
                );
            }
            Map<Long, PetriObjectStatisticsDTO> petriObjectsStatistics = new HashMap<>();
            for (List<Long> petriObjectIds : futures.keySet()) {
                Future future = futures.get(petriObjectIds);
                PetriObjectStatisticsDTO petriObjectStatisticsDTO;
                try {
                    petriObjectStatisticsDTO = (PetriObjectStatisticsDTO) future.get();
                } catch (Exception ex) {
                    throw new ParallelExecutionException(ex.getMessage());
                }
                for (long petriObjectId : petriObjectIds) {
                    List<Long> statisticsPetriObjectPlacesIds = petriObjectModelDTO.getPetriObjects().stream()
                            .filter(petriObjectDTO -> petriObjectDTO.getPetriObjectId() == petriObjectId)
                            .flatMap(petriObjectDTO -> petriObjectDTO.getPlaces().stream())
                            .filter(PlaceDTO::isCollectStatistics)
                            .map(PlaceDTO::getPlaceId)
                            .collect(Collectors.toList());
                    Map<Long, PlaceStatisticsDTO> placeStatisticsDTOs = new HashMap<>();
                    for (long statisticsPetriObjectPlacesId : statisticsPetriObjectPlacesIds) {
                        PlaceStatisticsDTO placeStatisticsDTO = petriObjectStatisticsDTO.getPlacesStatistics().get(statisticsPetriObjectPlacesId);
                        placeStatisticsDTOs.put(statisticsPetriObjectPlacesId, placeStatisticsDTO);
                    }
                    List<Long> statisticsPetriObjectTransitionIds = petriObjectModelDTO.getPetriObjects().stream()
                            .filter(petriObjectDTO -> petriObjectDTO.getPetriObjectId() == petriObjectId)
                            .flatMap(petriObjectDTO -> petriObjectDTO.getTransitions().stream())
                            .filter(TransitionDTO::isCollectStatistics)
                            .map(TransitionDTO::getTransitionId)
                            .collect(Collectors.toList());
                    Map<Long, TransitionStatisticsDTO> transitionStatisticsDTOs = new HashMap<>();
                    for (long statisticsPetriObjectTransitionId : statisticsPetriObjectTransitionIds) {
                        TransitionStatisticsDTO transitionStatisticsDTO = petriObjectStatisticsDTO.getTransitionsStatistics().get(statisticsPetriObjectTransitionId);
                        transitionStatisticsDTOs.put(statisticsPetriObjectTransitionId, transitionStatisticsDTO);
                    }
                    PetriObjectStatisticsDTO postprocessedPetriObjectStatisticsDTO = new PetriObjectStatisticsDTO(transitionStatisticsDTOs, placeStatisticsDTOs);
                    petriObjectsStatistics.put(petriObjectId, postprocessedPetriObjectStatisticsDTO);
                }
            }
            return new PetriObjectModelStatisticsDTO(petriObjectsStatistics);
        }
        private String getQueueName() {
            return queuesNames[queueNamesCursor.getAndIncrement() % queuesNames.length];
        }
    }
```

PetriObjectGraphVertex.java

```
package edu.pedorenko.petri.util.cycles_resolver;
import edu.pedorenko.petri.model.PetriObject;
import java.util.ArrayList;
import java.util.List;
class PetriObjectGraphVertex {
	private PetriObject petriObject;
	private List<PetriObjectGraphVertex> children = new ArrayList<>();
	private List<PetriObjectGraphVertex> reversedChildren = new ArrayList<>();
	PetriObjectGraphVertex(PetriObject petriObject) {
		this.petriObject = petriObject;
	}
	void addChild(PetriObjectGraphVertex petriObjectGraphVertex) {
		children.add(petriObjectGraphVertex);
	}
	void addReversedChild(PetriObjectGraphVertex petriObjectGraphVertex) {
		reversedChildren.add(petriObjectGraphVertex);
	}
	PetriObject getPetriObject() {
		return petriObject;
	}
	List<PetriObjectGraphVertex> getChildren() {
		return children;
	}
	List<PetriObjectGraphVertex> getReversedChildren() {
		return reversedChildren;
	}
}
```

StronglyConnectedComponentsResolver.java

```
package edu.pedorenko.petri.util.cycles_resolver;
import edu.pedorenko.petri.model.PetriObject;
import edu.pedorenko.petri.model.PetriObjectModel;
import edu.pedorenko.petri.model.PetriObjectModelException;
import edu.pedorenko.petri.model.arc.ArcIn;
import edu.pedorenko.petri.model.arc.ArcOut;
import edu.pedorenko.petri.model.transition.Transition;
import edu.pedorenko.petri.util.petri_object_merger.PetriObjectMerger;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.Stack;
import java.util.stream.Collectors;
public class StronglyConnectedComponentsResolver {
	private StronglyConnectedComponentsResolver() {
	}
	public static PetriObjectModel resolveStronglyConnectedComponents(PetriObjectModel petriObjectModel) throws PetriObjectModelException {
		List<List<PetriObjectGraphVertex>> stronglyConnectedComponents = findStronglyConnectedComponents(petriObjectModel);
		return createNewPetriObjectModel(stronglyConnectedComponents);
	}
	private static List<List<PetriObjectGraphVertex>> findStronglyConnectedComponents(PetriObjectModel petriObjectModel) {
		List<PetriObjectGraphVertex> graph = createGraph(petriObjectModel);
		List<List<PetriObjectGraphVertex>> stronglyConnectedComponents = new ArrayList<>();
		Set<PetriObjectGraphVertex> visitedVertex1 = new HashSet<>();
		Stack<PetriObjectGraphVertex> finishedVertex = new Stack<>();
		for (PetriObjectGraphVertex vertex : graph) {
			if (!visitedVertex1.contains(vertex)) {
				fillVisitedVertexStack(vertex, visitedVertex1, finishedVertex);
			}
```

```
        }
        Set<PetriObjectGraphVertex> visitedVertex2 = new HashSet<>();
        while (finishedVertex.size() > 0) {
            PetriObjectGraphVertex vertex = finishedVertex.pop();
            if (!visitedVertex2.contains(vertex)) {
                List<PetriObjectGraphVertex> stronglyConnectedComponent = new ArrayList<>();
                fillStronglyConnectedComponent(vertex, visitedVertex2, stronglyConnectedComponent);
                stronglyConnectedComponents.add(stronglyConnectedComponent);
            }
        }
        return stronglyConnectedComponents;
    }
    private static void fillVisitedVertexStack(
            PetriObjectGraphVertex vertex,
            Set<PetriObjectGraphVertex> visitedVertex,
            Stack<PetriObjectGraphVertex> finishedVertex) {
        visitedVertex.add(vertex);
        for (PetriObjectGraphVertex child : vertex.getChildren()) {
            if (!visitedVertex.contains(child)) {
                fillVisitedVertexStack(child, visitedVertex, finishedVertex);
            }
        }
        finishedVertex.push(vertex);
    }
    private static void fillStronglyConnectedComponent(
            PetriObjectGraphVertex vertex,
            Set<PetriObjectGraphVertex> visitedVertex,
            List<PetriObjectGraphVertex> stronglyConnectedComponent) {
        visitedVertex.add(vertex);
        stronglyConnectedComponent.add(vertex);
        for (PetriObjectGraphVertex reversedChild : vertex.getReversedChildren()) {
            if (!visitedVertex.contains(reversedChild)) {
                fillStronglyConnectedComponent(reversedChild, visitedVertex, stronglyConnectedComponent);
            }
        }
    }
    private static List<PetriObjectGraphVertex> createGraph(PetriObjectModel petriObjectModel) {
        List<PetriObjectGraphVertex> graph = new ArrayList<>();
        Map<PetriObject, PetriObjectGraphVertex> petriObjectToVertexMap = new HashMap<>();
        for (PetriObject petriObject : petriObjectModel) {
            PetriObjectGraphVertex petriObjectGraphVertex = new PetriObjectGraphVertex(petriObject);
            graph.add(petriObjectGraphVertex);
            petriObjectToVertexMap.put(petriObject, petriObjectGraphVertex);
        }
        for (PetriObjectGraphVertex petriObjectGraphVertex : graph) {
            PetriObject petriObject = petriObjectGraphVertex.getPetriObject();
            for (Transition transition : petriObject) {
                for (ArcOut arcOut : transition.getArcsOut()) {
                    PetriObject placePetriObject = arcOut.getPlace().getPetriObject();
                    if (petriObject != placePetriObject) {
                        PetriObjectGraphVertex child = petriObjectToVertexMap.get(placePetriObject);
                        petriObjectGraphVertex.addChild(child);
                        child.addReversedChild(petriObjectGraphVertex);
                    }
                }
                for (ArcIn arcIn : transition.getArcsIn()) {
                    PetriObject placePetriObject = arcIn.getPlace().getPetriObject();
                    if (petriObject != placePetriObject) {
                        PetriObjectGraphVertex child = petriObjectToVertexMap.get(placePetriObject);
                        petriObjectGraphVertex.addChild(child);
                        child.addReversedChild(petriObjectGraphVertex);
                        child.addChild(petriObjectGraphVertex);//common places are interpreted as cycles
                        petriObjectGraphVertex.addReversedChild(child);
                    }
                }
```

```
                }
            }
            return graph;
        }
        private static PetriObjectModel createNewPetriObjectModel(List<List<PetriObjectGraphVertex>> stronglyConnectedComponents) throws
PetriObjectModelException {
            List<PetriObject> petriObjects = new ArrayList<>();
            for (List<PetriObjectGraphVertex> stronglyConnectedComponent : stronglyConnectedComponents) {
                List<PetriObject> stronglyConnectedPetriObjects = stronglyConnectedComponent.stream()
                        .map(PetriObjectGraphVertex::getPetriObject)
                        .collect(Collectors.toList());
                if (stronglyConnectedPetriObjects.size() > 1) {
                    PetriObject mergedPetriObject = PetriObjectMerger.mergePetriObjects(stronglyConnectedPetriObjects);
                    petriObjects.add(mergedPetriObject);
                } else {
                    petriObjects.add(stronglyConnectedPetriObjects.get(0));
                }
            }
            return new PetriObjectModel(petriObjects);
        }
    }
```

PetriObjectMerger.java

```
package edu.pedorenko.petri.util.petri_object_merger;
import edu.pedorenko.petri.model.PetriObject;
import edu.pedorenko.petri.model.PetriObjectModelException;
import edu.pedorenko.petri.model.arc.ArcIn;
import edu.pedorenko.petri.model.arc.ArcOut;
import edu.pedorenko.petri.model.arc.InformationalArcIn;
import edu.pedorenko.petri.model.place.Place;
import edu.pedorenko.petri.model.transition.Transition;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
public class PetriObjectMerger {
    private PetriObjectMerger() {
    }
    public static PetriObject mergePetriObjects(List<PetriObject> petriObjects) throws PetriObjectModelException {
        PetriObject firstPetriObject = petriObjects.get(0);
        List<Long> petriObjectIds = new ArrayList<>(firstPetriObject.getPetriObjectIds());
        StringBuilder petriObjectNameSB = new StringBuilder(firstPetriObject.getPetriObjectName());
        List<Transition> transitions = new ArrayList<>();
        int priority = firstPetriObject.getPriority();
        int externalBufferSizeLimit = firstPetriObject.getExternalBufferSizeLimit();
        List<CreateArcInTask> createArcInTasks = new ArrayList<>();
        List<CreateArcOutTask> createArcOutTasks = new ArrayList<>();
        for (PetriObject petriObject : petriObjects) {
            for (Place place : petriObject.getThisObjectPlaces()) {
                place.resetPetriObject();
            }
        }
        addPetriObjectTransitions(firstPetriObject, transitions, createArcInTasks, createArcOutTasks);
        for (int i = 1; i < petriObjects.size(); ++i) {
            PetriObject petriObject = petriObjects.get(i);
            petriObjectIds.addAll(petriObject.getPetriObjectIds());
            petriObjectNameSB.append("+").append(petriObject.getPetriObjectName());
            if (priority < petriObject.getPriority()) {
                priority = petriObject.getPriority();
            }
            if (externalBufferSizeLimit > petriObject.getExternalBufferSizeLimit()) {
                externalBufferSizeLimit = petriObject.getExternalBufferSizeLimit();
            }
            addPetriObjectTransitions(petriObject, transitions, createArcInTasks, createArcOutTasks);
```

```
            }
            PetriObject mergedPetriObject = new PetriObject(
                        petriObjectIds,
                        petriObjectNameSB.toString(),
                        transitions,
                        priority,
                        externalBufferSizeLimit);
            createArcInTasks.forEach(CreateArcInTask::doTask);
            createArcOutTasks.forEach(CreateArcOutTask::doTask);
            return mergedPetriObject;
        }
        private static void addPetriObjectTransitions(
                    PetriObject petriObject,
                    List<Transition> transitions,
                    List<CreateArcInTask> createArcInTasks,
                    List<CreateArcOutTask> createArcOutTasks) {
            for (Transition transition : petriObject) {
                transition.resetPetriObject();
                Iterator<ArcIn> arcInIternator = transition.getArcsIn().iterator();
                while (arcInIternator.hasNext()) {
                    ArcIn arcIn = arcInIternator.next();
                    if (arcIn.getPlace().getPetriObject() != null) {
                        Place place = arcIn.getPlace();
                        int multiplicity = arcIn.getMultiplicity();
                        boolean isInformational = arcIn instanceof InformationalArcIn;
                        createArcInTasks.add(new CreateArcInTask(transition, place, multiplicity, isInformational));
                        arcInIternator.remove();
                    }
                }
                Iterator<ArcOut> arcOutIterator = transition.getArcsOut().iterator();
                while (arcOutIterator.hasNext()) {
                    ArcOut arcOut = arcOutIterator.next();
                    if (arcOut.getPlace().getPetriObject() != null) {
                        Place place = arcOut.getPlace();
                        int multiplicity = arcOut.getMultiplicity();
                        createArcOutTasks.add(new CreateArcOutTask(transition, place, multiplicity));
                        arcOutIterator.remove();
                    }
                }
                transitions.add(transition);
            }
        }
}
```

PetriNodeApplication.java

```
package edu.pedorenko.petrinode;
import org.modelmapper.ModelMapper;
import org.modelmapper.convention.MatchingStrategies;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
@SpringBootApplication
public class PetriNodeApplication {
    public static void main(String[] args) {
        SpringApplication.run(PetriNodeApplication.class, args);
    }
    @Bean
    public ModelMapper modelMapper() {
        ModelMapper modelMapper = new ModelMapper();
        modelMapper.getConfiguration().setMatchingStrategy(MatchingStrategies.STANDARD);
        return modelMapper;
    }
}
```

AmqpController.java

```
package edu.pedorenko.petrinode.controller;
import edu.pedorenko.petri.dto.PreprocessedPetriObjectDTO;
import edu.pedorenko.petri.dto.statistics.PetriObjectStatisticsDTO;
import edu.pedorenko.petrinode.model.computing_model.petri_object.parallel.ParallelComputingPetriObject;
import edu.pedorenko.petrinode.model.computing_model.petri_object.parallel.ParallelComputingPetriObjectModel;
import edu.pedorenko.petrinode.model.computing_model.time.TimeState;
import edu.pedorenko.petrinode.model.statistics.PetriObjectModelStatistics;
import edu.pedorenko.petrinode.model.statistics.PetriObjectStatistics;
import edu.pedorenko.petrinode.service.PetriObjectCreatingService;
import org.modelmapper.ModelMapper;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
@Component
public class AmqpController {
    private static final Logger logger = LoggerFactory.getLogger(AmqpController.class);
    private final PetriObjectCreatingService petriObjectCreatingService;
    private final ModelMapper modelMapper;
    private final ExecutorService executorService = Executors.newWorkStealingPool();
    @Autowired
    public AmqpController(
            PetriObjectCreatingService petriObjectCreatingService,
            ModelMapper modelMapper) {
        this.petriObjectCreatingService = petriObjectCreatingService;
        this.modelMapper = modelMapper;
    }
    @RabbitListener(queues = "#{environment.QUEUE_NAME}", concurrency = "100")
    public PetriObjectStatisticsDTO test(PreprocessedPetriObjectDTO petriObjectDTO) throws InterruptedException, ExecutionException {
        ParallelComputingPetriObjectModel petriObjectModel = petriObjectCreatingService.createPetriObjectModel(petriObjectDTO);
        ParallelComputingPetriObject parallelComputingPetriObject = petriObjectModel.getParallelComputingPetriObject(petriObjectDTO.getPetriObjectIds().get(0));
        parallelComputingPetriObject.setTimeState(new TimeState(petriObjectDTO.getTimeModelling()));
        Future future = executorService.submit(parallelComputingPetriObject);
        future.get();
        PetriObjectModelStatistics petriObjectModelStatistics = petriObjectModel.getPetriObjectModelStatistics();
        PetriObjectStatistics petriObjectStatisticsByPetriObjectId = petriObjectModelStatistics.getPetriObjectStatisticsByPetriObjectId(petriObjectDTO.getPetriObjectIds().get(0));
        PetriObjectStatisticsDTO petriObjectStatisticsDTO = convertToDTO(petriObjectStatisticsByPetriObjectId);
        System.out.println(petriObjectStatisticsDTO);
        return petriObjectStatisticsDTO;
    }
    private PetriObjectStatisticsDTO convertToDTO(PetriObjectStatistics petriObjectStatistics) {
        return modelMapper.map(petriObjectStatistics, PetriObjectStatisticsDTO.class);
    }
}
```

PetriObjectCreatingService.java

```
package edu.pedorenko.petrinode.service;
import edu.pedorenko.petri.dto.ArcDTO;
import edu.pedorenko.petri.dto.ArcInDTO;
import edu.pedorenko.petri.dto.DelayGeneratorDTO;
import edu.pedorenko.petri.dto.DelayGeneratorTypeDTO;
import edu.pedorenko.petri.dto.ExternalArcDTO;
import edu.pedorenko.petri.dto.PlaceDTO;
import edu.pedorenko.petri.dto.PreprocessedPetriObjectDTO;
```

```
import edu.pedorenko.petri.dto.TransitionDTO;
import edu.pedorenko.petrinode.exception.PetriObjectModelCreatingException;
import edu.pedorenko.petrinode.model.computing_model.petri_object.parallel.ParallelComputingPetriObjectModel;
import edu.pedorenko.petrinode.model.event_protocol.DummyEventProtocolFactory;
import edu.pedorenko.petrinode.model.model.PetriObject;
import edu.pedorenko.petrinode.model.model.PetriObjectModel;
import edu.pedorenko.petrinode.model.model.PetriObjectModelException;
import edu.pedorenko.petrinode.model.model.place.Place;
import edu.pedorenko.petrinode.model.model.transition.Transition;
import edu.pedorenko.petrinode.model.model.transition.delay_generator.ConstantDelayGenerator;
import edu.pedorenko.petrinode.model.model.transition.delay_generator.DelayGenerator;
import edu.pedorenko.petrinode.model.model.transition.delay_generator.ExponentialDelayGenerator;
import edu.pedorenko.petrinode.model.model.transition.delay_generator.NormalDelayGenerator;
import edu.pedorenko.petrinode.model.model.transition.delay_generator.UniformDelayGenerator;
import edu.pedorenko.petrinode.model.model_computer.PetriObjectModelComputer;
import org.springframework.stereotype.Service;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
@Service
public class PetriObjectCreatingService {
    public ParallelComputingPetriObjectModel createPetriObjectModel(PreprocessedPetriObjectDTO petriObjectDTO) {
        List<Transition> transitions = createTransitions(petriObjectDTO.getTransitions(), petriObjectDTO.getPlaces());
        List<PetriObject> petriObjects = new ArrayList<>();
        PetriObject requiredPetriObject;
        try {
            requiredPetriObject = new PetriObject(petriObjectDTO.getPetriObjectIds(), petriObjectDTO.getPetriObjectName(), transitions, 1,
Integer.MAX_VALUE);
            petriObjects.add(requiredPetriObject);
        } catch (PetriObjectModelException ex) {
            throw new PetriObjectModelCreatingException(ex.getMessage());
        }
        int transitionIdCounter = -1;
        int placeIdCounter = -1;
        try {
            for (ExternalArcDTO externalArc : petriObjectDTO.getExternalArcs()) {
                if (externalArc.getFromPetriObjectId() == petriObjectDTO.getPetriObjectIds().get(0)) {
                    PetriObject nextPetriObject = createNextPetriObject(externalArc.getToPetriObjectId(), externalArc.getToPlaceId(),
transitionIdCounter, placeIdCounter);
                    transitionIdCounter--;
                    placeIdCounter--;
                    petriObjects.add(nextPetriObject);

requiredPetriObject.getTransitionById(externalArc.getFromTransitionId()).addArcTo(nextPetriObject.getPlaceById(externalArc.getToPlaceId()));
                } else {
                    PetriObject prevPetriObject = createPrevPetriObject(externalArc.getFromPetriObjectId(), externalArc.getFromTransitionId(),
placeIdCounter);
                    petriObjects.add(prevPetriObject);

prevPetriObject.getTransitionById(externalArc.getFromTransitionId()).addArcTo(requiredPetriObject.getPlaceById(externalArc.getToPlaceId()));
                }
            }
            PetriObjectModel petriObjectModel = new PetriObjectModel(petriObjects);
            return PetriObjectModelComputer.createParallelComputingPetriObjectModel(petriObjectDTO.getPetriObjectIds().get(0), petriObjectModel, new
DummyEventProtocolFactory());
        } catch (PetriObjectModelException ex) {
            throw new PetriObjectModelCreatingException(ex.getMessage());
        }
    }
    private PetriObject createNextPetriObject(long toPetriObjectId, long toPlaceId, int transitionId, int placeId) throws PetriObjectModelException {
        Transition transition = new Transition(transitionId, "");
        transition.addArcFrom(new Place(toPlaceId, ""));
        transition.addArcTo(new Place(placeId, ""));
        return new PetriObject(toPetriObjectId, "", new ArrayList<Transition>() {{
```

```
            add(transition);
        }});
    }
    private PetriObject createPrevPetriObject(long fromPetriObjectId, long fromTransitionId, int placeId) throws PetriObjectModelException {
        Transition transition = new Transition(fromTransitionId, "");
        transition.addArcFrom(new Place(placeId, ""));
        return new PetriObject(fromPetriObjectId, "", new ArrayList<Transition>() {{
            add(transition);
        }});
    }
    private List<Transition> createTransitions(List<TransitionDTO> transitionDTOs, List<PlaceDTO> placeDTOs) {
        List<Transition> transitions = new ArrayList<>();
        Map<Long, Place> places = createPlaces(placeDTOs);
        for (TransitionDTO transitionDTO : transitionDTOs) {
            Transition transition = createTransition(transitionDTO, places);
            transitions.add(transition);
        }
        return transitions;
    }
    private Map<Long, Place> createPlaces(List<PlaceDTO> placeDTOs) {
        Map<Long, Place> places = new HashMap<>();
        for (PlaceDTO placeDTO : placeDTOs) {
            Place place = createPlace(placeDTO);
            if (places.containsKey(place.getPlaceId())) {
                throw new PetriObjectModelCreatingException("Duplicate place id \"" + place.getPlaceId() + "\"");
            }
            places.put(place.getPlaceId(), place);
        }
        return places;
    }
    private Place createPlace(PlaceDTO placeDTO) {
        return new Place(
                placeDTO.getPlaceId(),
                placeDTO.getPlaceName(),
                placeDTO.getMarking(),
                placeDTO.isCollectStatistics());
    }
    private Transition createTransition(TransitionDTO transitionDTO, Map<Long, Place> places) {
        DelayGenerator delayGenerator = createDelayGenerator(transitionDTO.getDelayGenerator());
        Transition transition = new Transition(
                transitionDTO.getTransitionId(),
                transitionDTO.getTransitionName(),
                transitionDTO.getPriority(),
                transitionDTO.getProbability(),
                delayGenerator,
                transitionDTO.isCollectStatistics());
        for (ArcInDTO arcDTO : transitionDTO.getArcsIn()) {
            Place fromPlace = places.get(arcDTO.getPlaceId());
            if (fromPlace == null) {
                throw new PetriObjectModelCreatingException(
                        "No place with id \"" + arcDTO.getPlaceId() + "\". " +
                                "Check arcs for transition with id \"" + transitionDTO.getTransitionId() + "\"");
            }
            if (arcDTO.isInformational()) {
                transition.addInformationalArcFrom(fromPlace, arcDTO.getMultiplicity());
            } else {
                transition.addArcFrom(fromPlace, arcDTO.getMultiplicity());
            }
        }
        for (ArcDTO arcDTO : transitionDTO.getArcsOut()) {
            Place toPlace = places.get(arcDTO.getPlaceId());
            if (toPlace == null) {
                throw new PetriObjectModelCreatingException(
                        "No place with id \"" + arcDTO.getPlaceId() + "\". " +
                                "Check arcs for transition with id \"" + transitionDTO.getTransitionId() + "\"");
```

```
                }
                transition.addArcTo(toPlace, arcDTO.getMultiplicity());
            }
            return transition;
        }
        private DelayGenerator createDelayGenerator(DelayGeneratorDTO delayGeneratorDTO) {
            DelayGeneratorTypeDTO delayGeneratorType = delayGeneratorDTO.getType();
            switch (delayGeneratorType) {
                case CONSTANT:
                    return new ConstantDelayGenerator(delayGeneratorDTO.getFirstParam());
                case UNIFORM:
                    return new UniformDelayGenerator(delayGeneratorDTO.getFirstParam(), delayGeneratorDTO.getSecondParam());
                case NORMAL:
                    return new NormalDelayGenerator(delayGeneratorDTO.getFirstParam(), delayGeneratorDTO.getSecondParam());
                case EXPONENTIAL:
                    return new ExponentialDelayGenerator(delayGeneratorDTO.getFirstParam());
                default:
                    throw new PetriObjectModelCreatingException("Unrecognized delay generator type: " + delayGeneratorType.name());
            }
        }
}
```

BeanUtil.java

```
package edu.pedorenko.petrinode.util;
import org.springframework.beans.BeansException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
import org.springframework.stereotype.Service;
@Service
public class BeanUtil implements ApplicationContextAware {
    private static ApplicationContext context;
    @Override
    public void setApplicationContext(ApplicationContext applicationContext) throws BeansException {
        context = applicationContext;
    }
    public static <T> T getBean(Class<T> beanClass) {
        return context.getBean(beanClass);
    }
}
```

DoubleToByteArrayUtil.java

```
package edu.pedorenko.petrinode.util;
import java.nio.ByteBuffer;
public class DoubleToByteArrayUtil {
    public static byte[] toByteArray(double value) {
        byte[] bytes = new byte[8];
        ByteBuffer.wrap(bytes).putDouble(value);
        return bytes;
    }
    public static double toDouble(byte[] bytes) {
        return ByteBuffer.wrap(bytes).getDouble();
    }}
```